UNIVERSITY OF ATHENS

MASTER THESIS

Approximating Minkowski Decomposition and 2D Subset Sum

Author:
Charilaos Tzovas

Supervisor:
Prof. Ioannis Z. Emiris

A thesis submitted in fulfillment of the requirements for the Master's Degree

in

$\mu\Pi\lambda\forall$

Department of Mathematics

June 10, 2016

UNIVERSITY OF ATHENS

Abstract

Faculty Name
Department of Mathematics

Master's Degree

Approximating Minkowski Decomposition and 2D Subset Sum

by Charilaos Tzovas

We consider the approximation of two NP-hard problems: Minkowski Decomposition (MinkDecomp) of lattice polygons in the plane and the closely related problem of Multidimensional Subset Sum (kD-SS) in arbitrary dimension. In kD-SS we are given an input set S of k-dimensional vectors, a target vector t and we ask if there exists a subset of S that sums to t. We prove, through a gap-preserving reduction, that, for general dimension k, kD-SS does not have a PTAS although the classic $1D$-SS does. On the positive side, we present an $O(n^3/\epsilon^2)$ approximation algorithm for $2D$-SS, where n is the cardinality of the set and ϵ bounds the difference of some measure of the input polygon and the sum of the output polygons. Applying this algorithm, and a transformation from MinkDecomp to $2D$-SS, we can approximate MinkDecomp. For an input polygon Q and parameter ϵ, we return two summands A and B such that $A + B = Q'$ with Q' being bounded in relation to Q in terms of volume, perimeter, or number of internal lattice points and an additive error linear in ϵ and up to quadratic in the diameter of Q. A similar function bounds the Hausdorff distance between Q and Q'. We offer experimental results based on our implementation which is openly provided via Github.

Acknowledgements

I want to thanks my family for their constant support all these years and my advisor for his help writing this thesis and during my Master's studies. All the people, professors and students alike, in the MPLA program, the members of the ERGA lab for all their help and comments and special thanks to Anna whom without I could not finish this project in time (if ever!)...

Contents

List of Figures

Chapter 1

Introduction

Factoring polynomials has a long history and is a fundamental tool in computer algebra systems. This, and other related concepts, are the main motivation for this thesis. We will briefly review some topics and approaches on this field and then present our own results and ideas. These ideas concern the decomposition of polygons that, through a theorem of Ostrowski, are related to the problem whether a polynomial is reducible or not. Most algorithms use tools (like GCD and lattice reductions) in order to find the coefficients of the factors. We will try to use a different approach that initiated mainly in Gao and Lauder [GL01] and Gao [Gao01]. It disregards the coefficients and uses the Newton polygon of a polynomial, namely, if its Newton polytope has a Minkowski decomposition. To deal with this NP-complete problem we propose an approximation algorithm. In this way, given a polygon Q, our approximation algorithm can return in polynomial time two polygons A and B that their Minkowski sum almost equals to Q. Apart from any application in factorization, the problem of Minkowski Decomposition of polygons (or polytopes) is significant on its own.

1.1 Historic overview

The topic of polynomial factorization is huge and has many different and interesting cases: univariate or multivariate, in which field are the coefficients ($\mathbb{Z}, \mathbb{Q}, \mathbb{F}_p$), whether the algorithms are deterministic or randomized. Here, we will just present the progress in the field mostly in the last 40 years. Most of the material of this section is from the surveys of Kaltofen [Kal85; Kal90; Kal92; Kal03], from the book of Gathen and Gerhard [GG03] and the references therein.

The first attempts on factorizing polynomials date back to Newton's Arithmetica Univesalis (1707) and later to the astronomer Friedrich T. v. Schubert who, in 1793, presented a finite step algorithm to compute the factors of a univariate polynomial with integer coefficients. In 1882 L. Kronecker rediscovered this method and generalized it to factor polynomials with two or more variables and coefficients in algebraic extensions. Another question is whether a polynomial can be factored or not. A criterion for deciding irreducibility was given by F. G. Eisenstein in 1850 (for the history of this theorem check [Cox11]) and later (around 1920) another criterion is attributed to A. Cohn [Mur02]. In recent years generalizations of Eisenstein's criterion appeared, for example [Bro08; KS97].

The first computer algebra systems that appeared around 1963-4 implemented the "almost a century old"-algorithm of Kronecker to factor a polynomial. The first improvement came with Berlekamp in 1967 [Ber67] with a deterministic algorithm

than can factor univariate polynomials in $\mathbb{Z}_p$, where p is prime, in time $O(n^3+prn^2)$ where n is the degree of the polynomial and r the number of actual roots. The Berlekamp algorithm takes as input a square-free polynomial $f(x)$ of degree n and returns a polynomial $g(x)$, with coefficients in the same field as $f(x)$, that divides $f(x)$. Applying the algorithm recursively in $g(x)$ eventually we will find a decomposition of $f(x)$ into irreducible polynomials. It was a great speed up compared to previous methods.

The application of the Hensel's lemma was introduced by Zassenhaus in 1969 [Zas69] in order to "lift" in k iterations a factorization modulo p to a factorization modulo p^{2^k}. From then on Hensel's lemma, or Hensel's lifting lemma, became a valuable tool in the field. Informally, the statement of the lemma says that if $f(x)$ is a polynomial with integer coefficients and $f(a)$ is "small" compared to $f'(a)$, then the equation $f(x) = 0$ has a solution near a. Then we can use the lemma to "lift" a root $r \mod p^k$ of $f(x)$ to a new root $r' \mod p^{k+1}$ such that $r \equiv r' \mod p^k$; we extend a factorization over $\mathbb{F}_p$ to one over $\mathbb{Z}/(p^k\mathbb{Z})$. In the coming years, from 1969 to 1976, many authors generalized and used Hensel's lifting lemma for factoring multivariate polynomials [Wan76; Tra76; WR76] or for multivariate GCD computations [MY73]. But even contemporary papers still use this powerful tool, for example [SGL04].

Around this time some randomized algorithm were proposed like the ones from Rabin and Berlekamp [RR79]. A landmark randomized algorithm is the Cantor-Zassenhaus algorithm that appeared in 1981 [DGC81] and can factor a polynomial $f(x) \in \mathbb{F}_p$ of degree n into irreducible factors in time $O(n^3 \log p)$. This algorithm, along with Berlekamp's algorithm, are widely used and implemented in most computer algebra systems.

Few years later another powerful algorithm made its appearance. The famous Lensta, Lenstra, Lovasz (LLL) algorithm was published in 1982 [LLL82] and it was the first polynomial time algorithm for factoring a univariate polynomial over $\mathbb{Q}$ or $\mathbb{Z}$. Since then it found many different applications. (Another problem that the LLL algorithm is applied is the *Shortest Vector Problem* (SPV). The problem consering us is a special case of the SPV and both will be formally defined later.) The LLL algorithm can decompose an integer polynomial of degree n into irreducible non-constant integer polynomials in time $O(n^4 \log B)$, where B bounds the lengths of the coefficients. At that time, it was also shown that factoring a dense multivariate polynomial can be reduced, in polynomial time, to univariate factoring [Kal82]. So, combining these two ideas, integer multivariate polynomials can be factored in polynomial time. At this point (we are around 1985), we can say that dense multivariate and univariate polynomials over the prime fields $\mathbb{F}_p$ and $\mathbb{Q}$ can be factored in polynomial time. But also, there are fields where the problem, even for univariate polynomials, is undecidable.

In 1992 Gathen and Shoup developed new techniques that allowed to implement the Cantor-Zassenhaus algorithm so that it uses an expected number of $O(n^{2+o(1)}+n^{1+o(1)} \log p)$ operations in $\mathbb{F}_p$ [GS92]. The first randomized subquadratic algorithm is due to Kaltofen and Shoup [KS98] that uses fast matrix multiplication and requires $O(n^{1.815} \log p)$ operation is $\mathbb{F}_p$. Two other randomized algorithm are of Gathen and Gao [GG94] and Kaltofen and Lobo [KL94]. Finding a deterministic algorithm that runs in polynomial time in n and $\log p$ (and not n and p) remains an

open problem.

In 2002, Hoeij described a new algorithm for factoring polynomials over $\mathbb{Q}$ by reducing the problem to an instance of Knapsack using power sums [Hoe02]. This results was generalized by Belabas [Bel04] to number fields but non of this these papers states a complexity bound. In [Bel+04] is it shown that this runs in polynomial time.

For the case of multivariate polynomials the Zassenhaus and LLL algorithms can be applied or Kaltofen's algorithm that reduces the problem to univariate factorization. Most recent improvements are the works of [SS93; NY02] and also in [GKL04]. In 1985, Gathen and Kaltofen [GK85] gave two polynomial time algorithms for factoring polynomials in two variables with coefficients in $\mathbb{F}_p$: the first algorithm is probabilistic with complexity $O(n \log p)^{O(1)}$ and the second algorithm is deterministic with complexity $O(dp)^{O(1)}$. The authors of [Bos+04] improved the complexity bounds by a deterministic algorithm with complexity $\widetilde{O}(n^{\omega+1})$ and a probabilistic algorithm with complexity $\widetilde{O}(n^{\omega})$ for factoring bivariate polynomials with coefficients in $\mathbb{F}_p$ ($2 \leq \omega \leq 3$ is the matrix multiplication exponent in $O(n^{\omega})$ with the current record being $\omega = 2.3728639$ by [CW90; Gal14]). Another approach of the problem is by Gao in [Gao03; Kal+08; SGL04] and additionally with a paper that is closely related to and motivated this work in [GL01] along with [ET06].

Many of the previous algorithms give exact solution to factorization while some of them offer approximate solution in the sense shown in the next example.

Example (of approximate factoring by Kaltofen). *Input polynomial to factor:* $81x^4 + 16y^4 - 648z^4 + 72x^2y^2 - 648x^2y^2 - 288y^2 + 1296 = 0$
Approximate solution: $(9x^2 + 4y^2 + 18\sqrt{2}z^2 - 36)(9x^2 + 4y^2 - 18\sqrt{2}2z^2 - 36) = 0$
that gives the polynomial:
$81x^4+16y^4-\underline{648.003z^4}+72x^2y^2+\underline{.002x^2z^2}+\underline{.001y^2z^2}-648x^2-288y^2-\underline{.007z^2}+1296 = 0$

Related to polynomial factorization is the problem of irreducibility testing, that is to test a given polynomial if it can be factored in a given field. Some algorithms use randomization to test irreducibility in the univariate case over a finite field like Rabin's algorithm in [RR79] that needs $O(n^2 \log n \log p)$ to test a polynomial of degree n over a field $\mathbb{F}_p$ (see also [GP97]). Polynomial time randomized algorithms also exist for the multivariate case over some field (one is [Gat85]). In [Kal87] Kaltofen presented an algorithm that tests dense multivariate polynomials over large finite fields for irreducibility in deterministic polynomial time. In [GL04] Gao and Lauder introduced an idea that does not involve computations in the given field by computations on the Newton polytope of the polynomial. Thus, it can test a polynomial for absolute irreducibility in any field, not necessarily computable.

Our work is mainly motivated by these last mentioned results of Gao and Lauder. In the papers [Gao01; GL01; GL04] the authors consider the Newton polytope of a polynomial and, using a theorem of Ostrowski, they can test if the polynomial is reducible or not exploiting the properties of its Newton polytope. Details on this approach will be given in the next chapters. So, we mostly attack the problem of Minkowski Decomposition of polygons by an approximation algorithm and try to use that to problems related to polynomials. In [SGL04; Sal04] they consider the problem of factoring a polynomial f knowing that is decomposable and given two summands that their addition given the Newton polytope pf

f. As it turns out this can be useful to us because our method always finds a decomposable polytope and two summands. Of course, apart from factorization and irreducibility testing, Minkowski decomposition is a natural problem and as such it may be useful in other applications. For example, other topics that the problem can find an application is Bezier patches [Gol03, chapter 8] or in implicitization where, for example, we want to construct matrices for the sparse resultant of 3 bivariate polynomials [KG03, section 10.3][EKZ15]. Or tropical geometry where the concept of Minkowski sum (and thus its inverse Minkowski decomposition) is fundamental [MS15].

1.2 Definitions-Preliminaries

We start with some very basic notions. We will give a simple definition of a polynomial: a univariate polynomial $f(x) \in \mathbb{F}$, where $\mathbb{F}$ is an arbitrary field, is an expression in the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

where x is the variable (or indeterminate) and the constants $a_i \in \mathbb{F}$. We can also use the notation

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

The **degree** $\deg f$ of a nonzero polynomial f is the largest n such that $a_n \neq 0$.

A multivariate polynomial with n variables $f(x_0, \ldots, x_n) \in \mathbb{F}$ can be written as

$$f(x_0, \ldots, x_n) = \sum a_{i_1 \ldots i_n} x_1^{i_1} \ldots x_n^{i_n}$$

In the present work we will only deal with bivariate polynomials. In this case a polynomial f usually is written as

$$f(x, y) = \sum_{i,j} a_{ij} x^i y^j$$

The degree of $f(x, y)$ with respect to variable x is $\deg_x f$ (or $\deg_y f$ respectively) and is the degree of the polynomial if we consider f as a univariate polynomial in x and y as part of the coefficients. The **total degree** of a monomial $x^i y^j$ is $i + j$ and the total degree $\deg f$ of the bivariate polynomial f is the maximum total degree of its monomials.

Let $f(x, y) = \sum a_{ij} x^i y^j$ be a bivariate polynomial. For every term with $a_{ij} \neq 0$, the corresponding exponent vector (i, j) viewed in $\mathbb{Z}^2$, is called a **support vector** of f. Define as $Supp(f)$ the set of all supporting vectors of f

$$Supp(f) = \{(i, j) \mid a_{ij} \neq 0\}$$

The **Newton polygon** (or polytope for higher dimensions) P_f of a bivariate

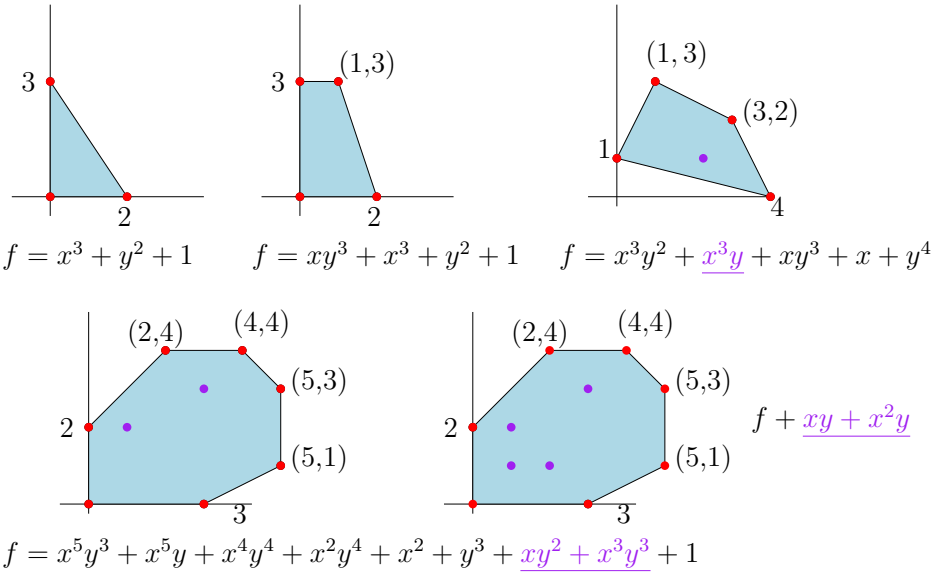

FIGURE 1.1: Examples of polynomials and their Newton polygons. In the two last examples some monomials are interior points of P_f.

polynomial f is the convex hull of $Supp(F)$. Sometimes we will also use the notation $NP(f)$ to denote the Newton polygon of f,

$$NP(f) = P_f = ConvHull(Supp(f))$$

Some examples can be seen in figure 1.1. Since coefficients are disregarded all example polynomials will have coefficients either 1 or 0 for the terms missing. An equivalent statement is that we consider bivariate polynomials over $GF(2)$.

A polygon P is called **integral** or **lattice polygon** when all its vertices are points with integer coordinates. All Newton polygons are integral polygons and every integer point corresponds to a monomial of the polynomial.

For every polynomial $f(x, y)$ there is a corresponding Newton polygon P_f, while a specific polygon can be generated by a number of different polynomials. For a given integral polygon $P \subset \mathbb{Z}^2$ we denote the family of polynomials that generate this polygon with $NP^{-1}(P)$:

$$NP^{-1}(P) = \{f(x, y) \mid NP(f) = P\}$$

Definition 1. The Minkowski sum (or addition) of two sets of vectors A and B in Euclidean space is defined by adding each vector in A to each vector in B, namely: $A + B = \{a + b \,|\, a \in A, b \in B\}$.

In figure 1.2 you can see two examples of the Minkowski addition of two polygons.

The main problem related to this work is the following.

Problem 2. Minkowski Decomposition (MinkDecomp)
Given a lattice convex polygon P, decide if it is decomposable, that is, if there are nontrivial lattice polygons A and B such that $A + B = Q$, where $+$ denotes Minkowski addition. The polygons A and B are called the *summands*.

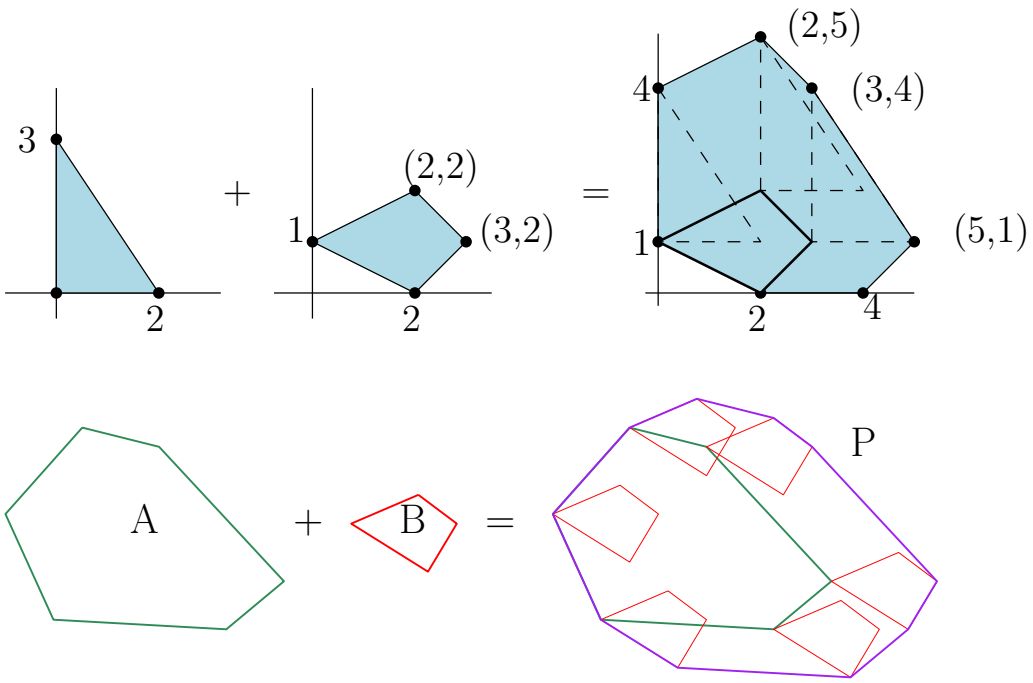

FIGURE 1.2: Two examples of the minkowski sum of polygons.

The decision version of problem 2 (whether a polygons admits a Minkowski decomposition) is proven NP-complete even for dimension 2 in [GL01] using a reduction from the Partition problem. Also in [GL01] the authors propose an algorithm to solve the problem in pseudo-polynomial time depending on the number of vertices and the maximum length of the edges. In [ET06] a different reduction from Subset-Sum is given and an improved algorithm is presented with optimum running time. For the reduction and more detailed analysis see section 3.2.

Our approach is motivated by Emiris and Tsigaridas [ET06] in the sense that we will use the reduction from Subset Sum to solve the MinkDecomp problem with an approximation algorithm. For this task we shall define an approximation version of the problem.

Problem 3. MinkDecomp-μ-approx

Input: A lattice polygon P, a parameter $0 < \epsilon < 1$ and a function μ.

Output: Lattice polygons A, B such that $0 \leq \mu(A+B) - \mu(P) < \epsilon \cdot \phi(D)$, where D is the diameter of P and $\phi()$ a polynomial. We call such an output an $\epsilon \cdot \phi(D)$-solution.

For μ we will consider the functions $vol(P)$: the volume, $per(P)$: the perimeter and $latt\_points(P)$: the internal lattice points of P ($\#(P \cap \mathbb{Z})$ but also we consider $d_H(P, A+B)$: the Hausdorff distance between P and $A+B$.

Let X and Y be two non-empty subsets of a metric space (M, d). We define their Hausdorff distance $d_H(X, Y)$ by

$$d_H(X,Y) = \max\{ \sup_{x \in X} \inf_{y \in Y} d(x,y), \sup_{y \in Y} \inf_{x \in X} d(x,y) \},$$

where *sup* represents the supremum and *inf* the infimum. In words, it is the longest minimum distance from one set to the other. It is defined for point sets but can be generalized for curves or surfaces by interpreting them as point sets.

A question is what bounds can we prove for other functions used to measure the similarity of two polygons like Fréchet distance or turning functions. It turns out that shape comparison is a big topic. For more information in shape matching and comparison see [Ark+91; Vel01; VH01; McC08].

Now, the missing connection is a theorem by Ostrowski that links polynomials and polytopes.

Theorem 4 (Ostrowksi [Ost76]). *Let $f, g, h \in K[x_1, x_2, \ldots, x_n]$ and P_f, P_g, P_h be their respective Newton polytopes in $\mathbb{R}^n$. If $f = gh$ then $P_f = P_g + P_h$.*

The contrapositive of this theorem gives at once an irreducibility criterion,

Corollary 5 ([GL04, cor. 2]). *Let $f \in K[x_1, x_2, \ldots, x_n]$ and P_f be its Newton polytope in $\mathbb{Z}^n$. If P_f is indecomposable and f is not divisible by any nonconstant monomial, then f is absolutely irreducible over K.*

Here absolute irreducibility means that the polynomial f is irreducible over the algebraic closure of K. A polygon P is *decomposable* if there exist non-trivial lattice polytopes A and B such that $A + B = P$ and by *non-trivial* we mean they cannot be points. A point as Newton polytope corresponds to a single monomial. Thus one method of detecting absolutely irreducibility of polynomials is to check whether their Newton polytopes are indecomposable. This is one possible application of our approach on MinkDecomp.

In section 3.2 we will present a reduction from MinkDecomp to the $2D$ Subset Set problem. Here we will only just give the definition of the problem for k dimensions and all further details are in chapter 3.

Problem 6. **kD-Subset Sum (kD-SS)**

Input: A vector set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n, k \geq 1\}$ and a target vector $t \in \mathbb{Z}^k$.

Output: Decide whether there exists a vector subset $S' \subseteq S$ such that $\sum v_i = t$, $v_i \in S'$.

This is a generalization of the classical Subset Sum problem and as such it is NP-complete.

1.3 Contribution - Organization

We introduce the kD-SS problem. It is clearly NP-complete and we prove that it cannot be approximated efficiently: for general dimension k it does not have a PTAS (although the classic 1D-SS has). In the coming sections we propose two algorithms to solve $2D$-SS approximately. Using this algorithms and the reduction from MinkDecomp, we can use them to provide approximate solution to the MinkDecomp problem. We prove that this solutions are bounded by some functions on the polygons, like volume and perimeter, but also from the Hausdorff

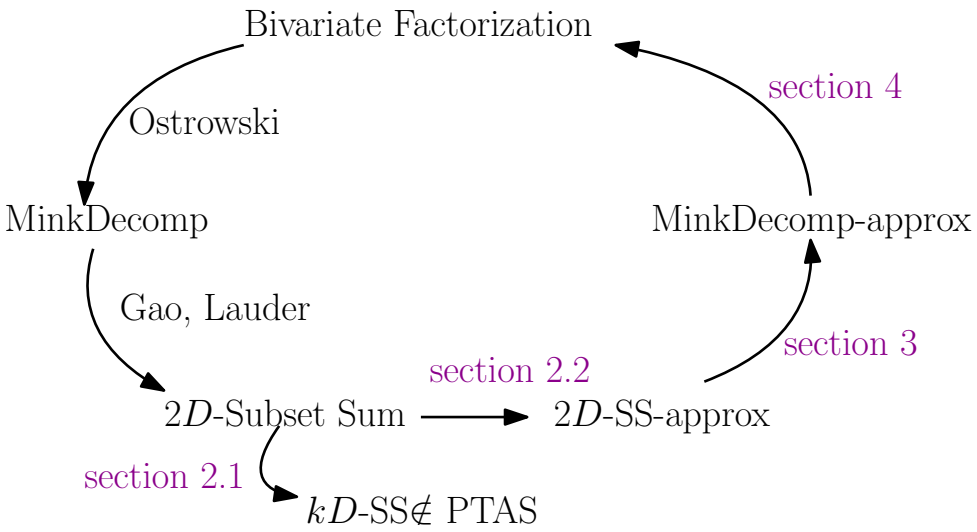

FIGURE 1.3: A sketch of the motivation and organization of this thesis.

distance. This is an approach to approximately decompose polygons. It is an subject of its own interest but our ultimate goal is to use this ideas on polynomials.

In the next chapter 2 we discuss the kD-SS problem. We present an inapproximability result but also polynomial time algorithms that provide some kind of approximation solutions. In 3 we use these algorithms to solve approximately the MinkDecomp problem for polygons

Chapter 2

Multi-dimensional Subset Sum

The classic Subset Sum (SS or $1D$-SS) problem is well studied and has many applications mainly in cryptography. In Subset Sum we are given a set of numbers S in $\mathbb{Z}$, a number $t \in \mathbb{Z}$ and we ask if there exist a subset of S such that its elements sum to t. This is a special case of the more general Knapsack problem that also appears in cryptography frequently and elsewhere. Both problem are standard NP-complete problems. We define a general version of Subset Sum where instead of numbers we are given k-dimensional vectors.

Problem 7. **kD-Subset Sum (kD-SS)**
Input: A vector set $S = \{v_i \mid v_i \in \mathbb{Z}^k, 1 \leq i \leq n, k \geq 1\}$ and a target vector $t \in \mathbb{Z}^k$.
Output: Decide whether there exists a vector subset $S' \subseteq S$ such that $\sum v_i = t$, $v_i \in S'$.

This is a generalization of the classical Subset Sum problem and as such it is NP-complete. Suppose the vectors in S are sorted in increasing vector length order and let P_i be the set of all possible vector sums that can be produced by adding vectors from the first i vectors in S. Then, $P_n \subseteq Z_k$ is the set of all possible vector sums we can produce by adding vectors from S.

Although $1D$-SS and kD-SS are NP-complete they are not strongly NP-complete and both can be solved exactly is pseudo-polynomial time: $1D$-SS is solved in $O(n|t|)$ [Cor+09] and, generalizing this idea, kD-SS is solved in $O(n|M|^k)$ [information in Zir14], where $M = \max P_n$ is the longest possible vector and $|\cdot|$ the euclidean norm.

Given the same input, another question is to find the longest and shortest vector in P_n. It seems that the longest vector problem can be solved in $O(n \log n)$ or $O(n)$ if vectors are sorted. Probably finding the shortest vector is harder, as is the shortest vector problem in lattices.

We care about an approximation approach and a fundamental concept are the approximation schemes.

Definition 8. A PTAS (Polynomial Time Approximation Scheme) is an algorithm which takes an instance of an optimization problem and a parameter $\epsilon > 0$ and, in polynomial time, produces a solution that is within a factor $1 + \epsilon$ of being optimal for minimization problems, or $1 - \epsilon$ for maximization problems.

One further defines the classes EPTAS (Efficient PTAS) where time complexity is polynomial in the input size (but can have any dependence on ϵ) and FPTAS (Fully PTAS) where the time complexity is polynomial in both input size and parameter ϵ.

We also say that a problem is in the complexity class PTAS (or just in PTAS) if there is a PTAS for the problem. The same applies for EPTAS and FTPAS. The class APX contains every problem that can be approximated within a constant factor c.

$1D$-SS is proven to be in FPTAS [IK75] when all numbers are positive (probably not the case when we allow negative numbers) and improved algorithm are given in [KPS97b; KPS97a]. Some related generalizations of the SS problem include the Multiple SS Problem (MSSP) which is a special case of the Multiple Knapsack Problem: MSSP is proven not to be in FPTAS but it admits a PTAS [CKP98]. In MSSP we are given multiple bins and we want to maximize the sum of the of the items in each bin without exceeding its capacity.

Another, closely related problem, is the Multidimensional Knapsack Problem (MKP) where each item has a k dimensional weight vector, a value and we must maximize the value of the selected items that can fit in the multidimensional knapasck. Firstly, this problem was proven not to be in FPTAS in [MC84; KPP04] and later this result was strengthened in [KS10] where the authors prove that the problem has no EPTAS. In the meantime it was known that MKP is in PTAS and such algorithms appeared in [Cap+00; FC84]. On the other hand, for $k = 1$, the classic Knapsack is in FPTAS [Vaz01, chap. 8].

Although $1D$-SS is a special case of the Knapsack problem it does not seem that this is the case for higher dimensions. The definition of kD-SS-opt given here makes the problem a minimization one while MKP is a maximization problem. The optimum values are different in each case and probably this makes kD-SS-opt harder to approximate.

Something similar holds for the SS problem. While $1D$-SS is in FPTAS we will prove that kD-SS cannot be approximated with any constant factor (is not in APX).

A similar problem to kD-SS is the well studied Closest Vector Problem (CVP): we are given a set of basis vectors $B = \{b_1, \ldots, b_n\}$, where $b_i \in \mathbb{Z}^k$, and a target vector $t \in \mathbb{Z}^k$, and we ask what is the closest vector to t in the lattice $\mathcal{L}(B)$ generated by B. This is $\mathcal{L}(B) = \{\sum_{i=1}^{m} a_i b_i \mid a_i \in \mathbb{Z}\}$ and thus kD-SS is a special case of CVP where $a_i \in \{0, 1\}$ but we do not know if it easier to solve. CVP is not in APX and cannot even be approximated within a factor of $2^{\log^{1-\epsilon} n}$ with $\epsilon = (\log(\log n))^c$ for $c < 1/2$ [Aro+97; Din+03]. Less are known for the related Shortest Vector Problem where we do not have a target vector t but we try to find the shortest vector in $\mathcal{L}(B)$.

2.1 kD-SS is not in APX

First, we must define an optimization version of the problem since approximation schemes apply for optimization problems while kD-SS is a decision problem.

Problem 9. **kD-SS-opt**
Given the set S and target t, find a subset of vectors $S' \subseteq S$ such that $\sum_{v_i \in S'} v_i = t'$ and

$$\min dist(t, t')$$

We are simply looking for the vector sum closest to our target. The distance can be the euclidean (l_2) or any other distance function.

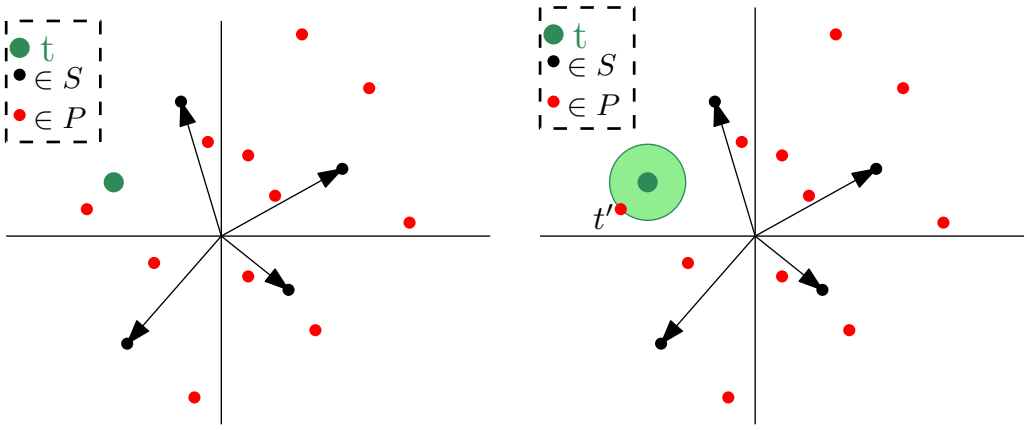

FIGURE 2.1: A decision and an optimization version of the problem.

For a different perspective, the question of the problem can also be stated as: find the nearest neighbour of t in P_n.

To prove that kD-SS-opt is not in APX we will apply the idea used to prove that CVP is not in APX [Aro+97]. We will change their proof and apply it to our problem to prove something similar for kD-SS-opt for general dimension k. Briefly, in SAT we are given a CNF formula ϕ and we want a valuation of its variables $x_1, \ldots, x_n$ that make ϕ true. In the Set Cover problem we are given a ground set of elements $\mathcal{U}$, the subsets $S_i \subset \mathcal{U},\ i = 1, \ldots, m$ and we want to find a collection of the sets S_i that cover all elements of $\mathcal{U}$.

Proposition 10. *[Bel+93] For every $c > 1$ there is a polynomial time reduction that, given an instance ϕ of SAT, produces an instance of Set Cover $\{\mathcal{U}, (S_1, \ldots, S_m)\}$ where $\mathcal{U}$ is the input set of integers and $S_1, \ldots, S_m$ are subsets of $\mathcal{U}$, and integer K with the following property: If ϕ is satisfiable, there is an exact cover of size K, otherwise all set covers have size more than cK.*

Given a CNF formula ϕ we invoke Proposition 10 and get an instance of the Set Cover problem. This is a gap introducing reduction, because if ϕ is satisfiable then the instance of Set Cover has a solution of size exactly K and if ϕ is not satisfiable every solution has size at least cK for a constant c. From this instance of Set Cover we create an instance for kD-SS that preserves the gap. Now, if ϕ is satisfiable, the closest vector to a given target t has distance exactly K. If ϕ is not satisfiable, the closest vector in target t has distance at least cK.

We reduce kD-SS to Set Cover for norm l_1, but this can easily be generalized to any l_p, where p is a positive integer since $l_p = (x_1^p + x_2^p + \cdots + x_n^p)^{1/p}$ for a vector $(x_1, x_2, \ldots, x_n)$. We say that a cover is *exact* if the sets in the cover are pairwise disjoint.

Theorem 11. *Given a CNF formula ϕ and $c > 1$ we create an instance $\{v_1, \ldots, v_m; t\}$ of kD-SS. If ϕ is satisfiable, then the minimum distance of a possible vector sum from t is smaller than K otherwise, it is larger than cK.*

Proof. Let $\{\mathcal{U}, (S_1, \ldots, S_m), K\}$ be the instance of Set-Cover obtained in proposition 10 for the formula ϕ. We transform it to an instance of kD-SS with input set

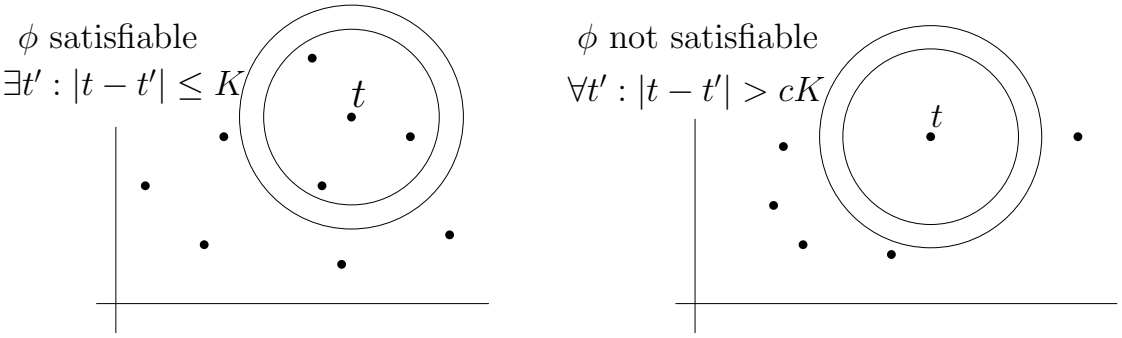

FIGURE 2.2: When ϕ is satisfiable there exists a vector sum close to the target. Otherwise, all vector sums are far.

$\{v_1, \ldots, v_m\}$ and target t, such that the distance of t from the nearest point in the set of all possible points P_n is either K or $\geq cK$.

Let $v_i \in \mathbb{Z}^{n+m}$, where $|\mathcal{U}| = n$. We will create such a vector v_i for every set S_i; $1 \leq i \leq m$. Let $L = cK$. Then, the first n coordinates of each vector v_i have their j'th-coordinate ($j \leq n$) equal to L if the corresponding j'th-element belongs to set S_i, or 0 otherwise. The remaining m coordinates have 1 in the $(n+i)$'th-coordinate and zeros everywhere else:

$$v_i = (L \cdot \chi_{S_i}, 0, \ldots, 1, \ldots 0)$$

where χ_{S_i} is the characteristic function of the set S_i. The target vector t has in the first n coordinates L and the last m coordinates are zeros, $t = (L, \ldots, L, 0, \ldots, 0)$.

Now, let the instance of Set-Cover have an exact cover of size K. We will prove that the minimum distance of every $v \in P_n$ from target t is less than K. Without loss of generality, let the solution be $\{S_1, \ldots, S_K\}$. For each S_i, $1 \leq i \leq K$, sum the corresponding vectors v_i and let this sum be $\zeta \in \mathbb{Z}^{n+m}$:

$$\zeta = \sum_{i=1}^{K} v_i = (\underbrace{L, \ldots, L}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K}).$$

The first n coordinates must sum up to $(L, L \ldots, L)$, because if one of the coordinates was 0, the solution would not be a cover and if one of them was greater than L, then some element is covered more than once and the solution would not be exact. The key point is that in the last m coordinates we will have exactly K units and everything else 0 since the size of the cover is K. The distance of this vector ζ from t is

$$\| - t + \zeta\|_1 = \|(\underbrace{0, \ldots, 0}_{n}, \underbrace{1, \ldots, 1}_{K} \underbrace{0, \ldots, 0}_{m-K})\|_1 = K$$

Thus, there is a point in P_n that its distance from t is at most K. (Actually the exact cover has distance exactly K).

Let us consider the other direction, where the Set Cover instance has a solution set greater than $cK = L$. We will show that the closest vector to t has distance at least L from t. This solution must have at least $cK = L$ sets. As before, $\| - t + \zeta\|_1 \geq L$ (this time the cover need not be exact).

Towards a contradiction, suppose there exist sets $S_1, \ldots, S_L$ that the corresponding vectors sum to a vector ξ whose distance from t is less than L:

$$\xi = \sum_{i=1}^{L} v_i \text{ and } \| -t + \xi\|_1 < L$$

If the sets $S_1, \ldots, S_L$ do not form a cover of S then some element of $\mathcal{U}$ is not covered and one of the first n coordinates of ξ is 0 and this alone is enough for $\| -t+\xi\|_1 > L$. If the sets form a cover that is not exact, then one element is covered more than once and at least one of the first n coordinates of ξ will be greater than L and will force $\| -t+\xi\|_1$ to be greater than L. Finally, if the sets form an exact cover, the first n coordinates of $\| -t + \xi\|_1$ will be 0. For the distance to be less than L, in the last m coordinates there must be less than L units implying that the sets in the cover are less than L contradicting our hypothesis.

In all cases, there cannot exist a vector whose distance from t is $< cK$. □

Using this reduction we can state the next theorem.

Theorem 12. *There is no APX algorithm for kD-SS-opt unless P=NP.*

Proof. Let ϕ be a given formula as an instance of SAT. Use proposition 10 to get an instance of Set Cover and then the reduction from theorem 11 to get an instance of kD-SS. Suppose there exists an algorithm $\mathcal{A}$ for kD-SS-opt that is in APX. $\mathcal{A}$ returns a vector t' such that $\|t - t'\|_1 \leq (1 + \epsilon)OPT$, where $OPT = \|t - t^*\|_1$ and t^* is the closest vector in P_n. From theorem 11, if ϕ is satisfiable then $OPT \leq K$ and if ϕ is not satisfiable $OPT > cK$.

We must run algorithm $\mathcal{A}$ with a suitable parameter ϵ so we can distinguish if the optimum solution t^* is within distance K or not. When ϕ is satisfiable we would want $(1+\epsilon)K < cK \implies \epsilon < c-1$. Set $c' < c-1$, call $\mathcal{A}$ with parameter $\epsilon = c'$ and let t' be the returned vector. In the case where ϕ is satisfiable and $OPT \leq K$ we have

$$\|t - t'\|_1 \leq (1 + \epsilon)OPT < cK$$

Of course if ϕ is not satisfiable for any t' we have that $\|t - t'\|_1 > cK$. Thus, $\|t - t'\|_1 < cK$ if and only if ϕ is satisfiable. Since ϵ is a constant and $\mathcal{A}$ is in APX we can decide SAT in polynomial time. □

Although there can be no algorithm that returns an constant factor approximation solution for general dimension k, we will present algorithms that provide different kind of approximation. Specifically, the returned vector t' of our algorithm is an $(OPT + \epsilon M)$ solution where $M = \max P_n$ is the longest possible vector sum.

2.2 2D-SS approximation algorithm

The initial motivation was to extend the already known FPTAS algorithm for $1D$-SS in the two dimensional case and provide an FPTAS for $2D$-SS; partially this failed . We are able to provide another type of approximation if we do not impose any constrains to the input vectors. The FTPAS for SS restricts the input to positive

numbers and it can be found in many textbooks like [Cor+09, chap. 35]. We will briefly describe it here.

Problem 13. $1D$-SS-opt
Input: A set of n integers $S = \{a_i \mid a_i \in \mathbb{N}, i = 1, \ldots, n\}$, a parameter $0 \leq \epsilon \leq 1$ and a target $t \in \mathbb{N}$.
Find a subset $S' \subset S$ that its elements sum to t' and $\max(t')$ while $t' \leq t$.

The FPTAS for this problem creates the lists E_i with the intermediate sums and then trims each list by a parameter δ. That is, remove an element $v \in E_i$ if there $w \in E_i$ such that $w \leq v \leq (1+\delta)w$. For $\delta = 2\epsilon/2$ this algorithms returns a solution t' such that $t' = (1+\epsilon)OPT$ in time $O(n^2 \log t/\epsilon)$. We try to apply the same idea of trimming in the $2D$ case and see why this does not perform so well in that case.

Our input is the set $S = \{v_1, v_2, \ldots, v_n\}$ with $v_i = (x_i, y_i) \in \mathbb{Z}^2$ and target $t \in \mathbb{Z}^2$ and we want to find $S' \subset S$ such that the vectors in S' sum to t. Suppose that vectors in S are sorted in increasing order according to their lengths. Also, P_i is the set of all possible vectors that can be produced by adding at most i elements from the first i vectors in S. P_n is the set of all possible vector sums.

2.2.1 An exact and exponential time algorithm

First we will describe an exact and exponential time algorithm for $2D$-SS and then we will modify that to take the approximation version.

Set the list $E_0 = \{0\}$ and for all $1 \leq i \leq n$,

$$E_i = E_{i-1} \cup \{v_i + w \mid w \in E_{i-1}\}$$

and sort in increasing order.

The first few steps will give:
$E_0 = \{0\}$
$E_1 = \{0, v_1\}$
$E_2 = \{0, v_1, v_2, v_1 + v_2\}$
$E_3 = \{0, v_1, v_2, v_3, v_1 + v_2, v_1 + v_3, v_2 + v_3, v_1 + v_2 + v_3\}$
$\vdots$

At the end, $E_n = P_n$ and will contain all possible $O(2^n)$ vector sums. From the output E_n we can decide if there exist a vector sum equal t or find a t' that is closest to t.

2.2.2 2D-SS approx

Now, in order to achieve polynomial time, at each step we will trim the lists E_i using an appropriate factor δ, like the 1-dimensional FPTAS, and then add the next v_{i+1}. Let $E_i = L_{i-1} \cup \{w + v_i \mid w \in L_{i-1}\}$ be the list created at the beginning of every step and that is about to get trimmed, $L_i = trim(E_i, \delta)$ is the trimmed list and $0 \leq \delta \leq 1$. The factor δ plays a role in the running time, space and approximation ratio. It depends on ϵ and n and it will be set later.

At the beginning of the i-th step we create the list E_i from the previous trimmed list. Sort E_i based on the lengths and call $L_i = trim(E_i, \delta)$.In trim, for each vector

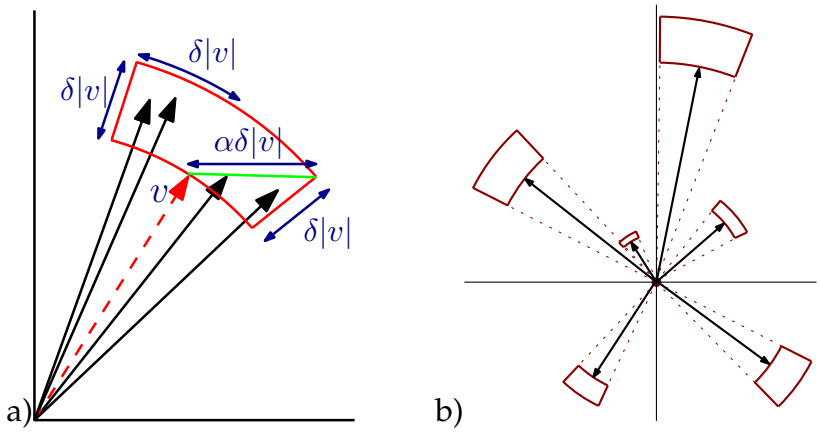

FIGURE 2.3: a) A single cell for the dashed vector v. All vectors in the cell will be deleted. The distances are shown and the furthest point is in distance $\alpha\delta|v|$. b) A few cells. The shorter the vector the smaller the cell.

$u \in E_i$ with length $|u|$ and angle $\theta(u)$ from the x-axis, first copy u in L_i. Then, from the vectors $u' \in E_i$, check only those u' that have length

$$|u| \leq |u'| \leq (1+\delta)|u| \tag{2.1}$$

If also

$$\theta(u) - \delta \leq \theta(u') \leq \theta(u) + \delta \tag{2.2}$$

remove u' from E_i.

This two conditions, eq. (2.1) eq. (2.2), ensure that $dist(u', u) \leq \alpha\delta|u|$, where $1 \leq \alpha \leq 2$ is a constant. For every vector in the original list E_i there is a vector in the trimmed list L_i that is not very far:

$$\forall u \in E_i \exists w \in L_i : u = w + r_w,\ |r_w| \leq \alpha\delta|w| \tag{2.3}$$

hence, $|w| \leq |u| \leq (1+\delta)|w|$ (see figure 2.3). Function *trim* can be seen in pseudocode in algorithm 2 and the whole algorithm in algorithm 1.

Since all vectors have integer coordinates, any vector $u \in E_i$ such that $|u| < 1/\alpha\delta$ implies that $\alpha\delta|u| < 1$. Thus, the area around u does not contain any other lattice points except u.

Algorithm 1: approx-2D-SS

input : $S \subset \mathbb{Z}^2,\ 0 \leq \epsilon \leq 1$
output: all approximation points L_n

L_0=∅
for $v_i \in S$ **do**
  $E_i = L_{i-1} \cup \{L_{i-1} + v_i\}$
  $L_i = \text{trim}(E_i, \epsilon/2n)$
return L_n

Using algorithm 2 as the main procedure we have algorithm 1 that provides approximate solutions to the $2D$-SS problem.

Algorithm 2: trim

input : $E \subset \mathbb{Z}^2, 0 \leq \delta \leq 1$
output: a trimmed list L

```
sort(E)
for v_k ∈ E do
    i = 1
    while |v_{k+i}| ≤ (1 + δ)|v_k| do
        if θ(v_{k+i}) − δ ≤ θ(v_k) ≤ θ(v_{k+i}) + δ then
            remove v_{k+i} from E
        i = i + 1
return E
```

We will bound the size of the lists L_i ($|L_i| = \Theta(|E_i|)$) in order to bound the total running time and the space requirements of the algorithm.

Lemma 14. *Using the above notation, call function L_i =trim(E_i, δ), with parameter $\delta = \epsilon/2n$. It holds that $|E_i| = O(n^2\epsilon^{-2} \log M_n)$ for $1 \leq i \leq n$.*

Proof. Let $M_i = max\{|u| : u \in E_i\}$, the vector in E_i with the largest magnitude. Every vector in E_i has length between $(1+\delta)^k$ and $(1+\delta)^{k+1}$. These are circles with center $(0,0)$ and radius $(1+\delta), (1+\delta)^2, \ldots, (1+\delta)^k$ for some k. Every two successive circles form an annulus that we call it a zone. We must cover all $u \in P_n$ and k is the minimum such that $(1+\delta)^k > M_n$. Solving $(1+\delta)^k \geq M_n$ for k, there are $O(n \log M_n/\epsilon) = O(n^2/\epsilon)$ many zones that can be created.

Every zone is divided into cells. Each cell is taken in such a way that it will cover $2\delta R$ of the inner circle of the zone, where R is the radius of this circle (figure 2.3a). Thus, every zone between the circles with radious R and $(1+\delta)R$ has at most $2\pi R/\delta R = 4\pi n/\epsilon$ cells.

Since a list L_i has at most an entry for every cell created in every zone, its size can be at most $(n \log M_n/\epsilon) \cdot (4\pi n/\epsilon) = O(n^2\epsilon^{-2} \log M_n)$. □

For function trim, the time required is $|E_i|$ to consider all vectors and, in the worst case, we have to check each vector in E_i with all the others leading to a running time of $T(trim) = O(|L_i|^2)$. The running time for algorithm 1 is $n \cdot T(trim) = O(n|L_n|^2)$ and overall, from lemma 14, it requires time $O(n^5\epsilon^{-4} \log^2 M_n)$. The algorithm only stores at each step the list L_i so the space consumption is $O(n^2\epsilon^{-2} \log M_n)$.

Corollary 15. *For $\delta = \epsilon/2n$, the running time of algorithm 1 is $O(n^5\epsilon^{-4} \log^2 M)$ and space required is $O(n^2\epsilon^{-2} \log M_n)$.*

Our next task is to bound the approximation ratio. The goal was an $(1+\epsilon)$ error but this is not the case for this algorithm. The actual bound we can prove is ϵM can be seen in Corollary 17. This is a quite relaxed result but we are not able to improve it at this time. As it seems, while for dimension 1 the problem is easier, the 2 dimensional case immediately gets harder. The key difference is

in the definitions of the problems. When we try to maximize the sum, in $1D$-SS, the optimum value is $\Theta(|t|)$ whereas, in the minimization version of $2D$-SS the optimum value can be arbitrary small, way smaller than any vector length. One can define a maximization version of the problem but this does not captures the intuition we would like. An optimum, maximum length vector t' can be opposite to target t and theit difference $|t - t'|$ can be even $2|t'|$.

Theorem 16. *For a set of vectors $S = \{v_i \mid v_i \in \mathbb{Z}^2,\ 0 \leq i \leq n\}$, every possible vector sum $v \in P_n$ can be approximated by another vector sum w:*

$$\forall v \in P_n \exists w \in L_n, \exists r_w \in \mathbb{Z}^2 : v = w + r_w,\ |r_w| \leq n\delta M_n$$

This also means that $dist(u, w) \leq n\delta M_n$.

Proof. The proof is by induction. For the base step, it is easy to see that if we only have one element the theorem holds. The induction hypothesis is:

$$\forall v \in P_{n-1} \exists w \in L_{n-1}, \exists r_w : v = w + r_w,\ |r_w| \leq (n-1)\delta M_{n-1}$$

Now suppose $v \in P_n \setminus P_{n-1}$, because if $v \in P_{n-1}$ the theorem holds straight from the induction hypothesis. We write v as $v = z + v_n,\ z \in P_{n-1}$ and the induction hypothesis holds for z, thus

$$\exists p \in L_{n-1} \exists r_p : z = p + r_p,\ |r_p| \leq (n-1)\delta M_{n-1}. \tag{2.4}$$

Since $p \in L_{n-1}$ this means that $p + v_n \in E_n$ and $L_n = trim(E_n)$. From the guarantee of function trim we know that

$$\exists q \in L_n : p + v_n = q + r_q,\ |r_q| \leq \delta|q|. \tag{2.5}$$

From eqs. (2.4) and (2.5) we get $v = z + v_n = p + v_n + r_p = q + r_q + r_p$. This proves that for $v \in P_n$ there exists a vector $q \in L_n$ that approximates it; but how close are they? We will bound the length $|r_q + r_p|$. From eqs. (2.4) and (2.5),

$$|r_p| \leq (n-1)\delta \max\{L_{n-1}\} \leq (n-1)\delta M_n$$
$$|r_q| \leq \delta|q|,\ q \in L_n \implies |r_q| \leq \delta M_n$$

Thus

$$|r_q + r_p| \leq |r_q| + |r_p| \leq (n-1)\delta M_n + \delta M_n \leq n\delta M.$$

□

Setting $\delta = \epsilon/2n$ we can ensure that every vector will be approximated by a vector in L_n at most ϵM_n far from the optimum. Again, $OPT = \min dist(t, P_n)$.

Corollary 17. *Given a vector $t \in \mathbb{Z}^2$, for $\delta = \epsilon/2n$ there is $w \in L_n$ such that $dist(t, w) \leq OPT + \epsilon M_n$.*

Proof. Let $v \in P_n$ be the vector sum that achieves the minimum distance from t, its nearest neighbour, and $OPT = dist(v, t)$. By theorem 16 and $\delta = \epsilon/2n$, exists $w \in L_n$ such that $dist(v, w) \leq \epsilon M_n$ thus $dist(t, w) \leq OPT + \epsilon M_n$. □

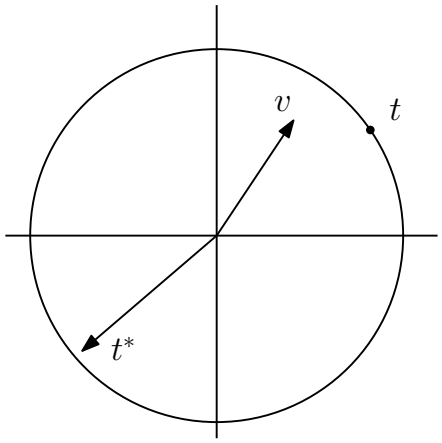

FIGURE 2.4: If we consider the maximization version of the problem, the optimum solution t^* does not seem as "optimum" as vector v.

To further illustrate the importance of how one defines the problem we will see what happens in a maximization version. Now we want the vector $v \in P_n$ that maximizes $|v|$ while $|v| \leq |t|$. This last condition can also be $v_x \leq t_x$ and $v_y \leq t_y$ but it is equivalent. Let the optimum vector that maximizes the length be t^*. See in figure 2.4 that this setting is not what we usually would prefer.

If we restrict our input to only positive vectors (actually we want all vectors to belong to one quadrant) and there are no short vectors, we can alter the proof of theorem 16 and prove that

$$\forall v \in P_n \, \exists w \in L_n, \, \exists r_w : v = w + r_w, \; |r_w| \leq n\delta|w|$$

which also implies that $|w| \leq |v| \leq (1+n\delta)|w|$. Under the maximization definition, and for $\delta = \epsilon/n$, this is a FPTAS algorithm because now $OPT = |v|$.

2.2.3 A grid based algorithm

It turns out that the same approximation ratio ϵM_n can be achieved by a faster algorithm that separates the plane into a grid. Instead of creating this different annulus-slice cells we have a regular orthogonal grid where each square cell has side $d = \epsilon M_n/2n$. Many thanks to Günter Rote for the fruitful conversation.

Again we construct the lists E_i and L_i but the trimming is different. This time every vector in a cell is represented by a vector in one of the corners of the cell. Let the cell side length be $d = \epsilon M_n/2n$, and for each $v(x, y) \in E_i$ store in the trimmed list L_i the vector with its coordinates rounded in the integer multiple of d:

$$\forall v(x, y) \in E_i \, \exists w(x', y') \in L_i : x' = \left\lfloor \frac{x}{d} \right\rfloor, \; y' = \left\lfloor \frac{y}{d} \right\rfloor$$

and

$$dist(v, w) \leq \sqrt{2}d \leq \frac{\epsilon M_n}{n}$$

and the maximum value reaches when v and w are in the diagonal of the cell.

The whole grid has size $2M_n$ and since $d = \epsilon M_n/2n$ the grid has $O(M_n/d) = O((n/\epsilon)^2)$ cells. In the worst case we will have a vector in every cell and this means that the time to traverse the lists E_i at each step is $O(n^2\epsilon^{-2})$. Since we have n lists

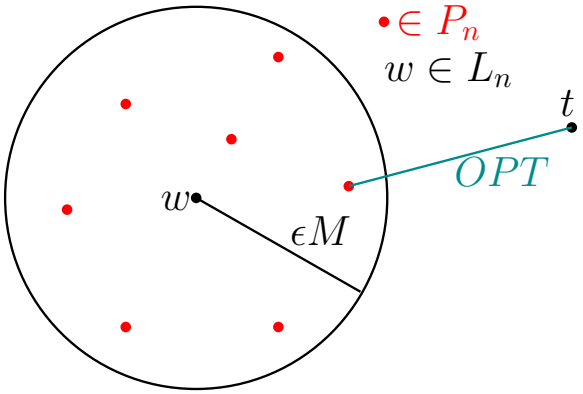

FIGURE 2.5: For every t the returned vector is at most ϵM from the optimum

the total running time of the new algorithm is $O(n^3\epsilon^{-2})$ and the space requirements are $O(n^2\epsilon^{-2})$.

Also, every $u \in P_n$ is the sum of at most n vectors from S. In the worst case we "lose" d every time we call trim. In that case we lost at most $nd = \epsilon M_n$:

$$\forall v \in P_n \exists w \in L_n : dist(v, w) \leq \epsilon M_n$$

A remark. There is a factor $\sqrt{2}$ that we also omit it from our approximation. This happens because the maximum distance inside a cell is not d but $\sqrt{2}d$. For dimension k the minimum distance is $\sqrt{k}d$ and this affects the algorithm in higher dimension (see section 2.3).

Corollary 18. *The grid-based algorithm runs in time $O(n^3\epsilon^{-2})$, requires space $O(n^2\epsilon^{-2})$ and returns a solution t' such that $dist(t, t') \leq OPT + \epsilon M_n$*

2.2.4 Combining the two ideas

The first algorithm seems to approximate intermediate results better. For short vectors that appear in some E_i trims them according to their length. But has to check the whole list E_i in the worst case in order to remove close vectors. This adds a quadratic factor to the running time. At this point we cannot exploit the "better" approximation behaviour and the bound proven seems quite loose.

The second, grid-based algorithm "cuts" a constant d from every factor regardless of its length. If a vector has length 10,10^3 or 10^5 the algorithm will behave the same. But it is faster because it does not have to check any other vector; for every vector it sees it rounds it on the spot thus having linear time to the size of the lists.

What if we can combine the better aspects of these two approaches? The better approximation behaviour of the first algorithm and the speed of the second. We will keep the ideas of the first and try to trim every vector "on each own", meaning that we will not check with the other vectors saving the quadratic factor (see figure 2.6).

At step i we have the list E_i and we will trim it with a factor δ. We will consider the polar coordinates of the vectors. For a vector $v(\phi, r)$ let $\phi = \theta(v)$ be the angle

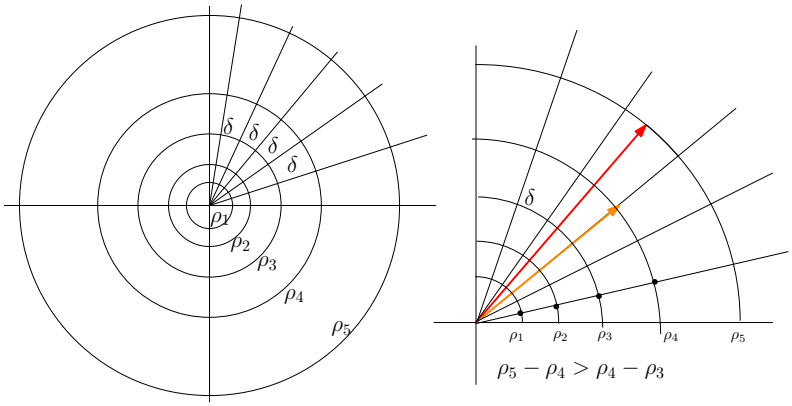

FIGURE 2.6: How space is divided. The red vector is deleted and replaces by the yellow one.

with the x axis and $r = |v|$ its euclidean length. Let v be a vector in E_i and we will change v to $v' = (\phi', r')$. First round its angle in multiple of δ:$\phi' = \lfloor \phi/\delta \rfloor$. Next we will round its length. The idea is to round in such a way that shorter vectors are approximated better than the long ones. Rounding to a multiple of some d does not provide that.

For rounding the lengths we will construct an array A with all the acceptable rounded lengths. The entries A are the lengths $[1, (1+\delta), \ldots, (1+\delta)^i]$ for the minimum i such that $(1+\delta)^i > M_n$. Solving this inequality, as in lemma 14, we get that $i = O(n \log M_n/\epsilon)$ and this is the size of A. Now, for a vector v we just make a binary search in A for $|v|$ that returns the zone such that $(1+\delta)^i \leq |v| \leq (1+\delta)^{i+1}$ and $r' = bin\_search(A, |v|) = (1+\delta)^i$.

The space is divided in $O(\delta) = O(e/\epsilon)$ angles and $O(n \log M_n/\epsilon)$ different lengths. Each E_i has size $O(n^2\epsilon^{-2} \log M_n)$; we save the quadratic factor but add a $\log |E_i|$ for the binary search. The whole algorithm runs in time

$$O(n|E_i| \log |E_i|) = O(n^3\epsilon^{-2} \log \frac{n \log M_n}{\epsilon}) = O(n^3\epsilon^{-2} \log \frac{n}{\epsilon})$$

and provides the same approximation since $\forall v \in E_i, \exists w \in L_i : dist(v, w) \leq \epsilon|w|$ as before.

So why all the fuzz? We just have a slower algorithm with the same bound. We believe that the better behaviour of this algorithm can be suited for an average case analysis that will prove that the algorithm provides a better approximation ratio for the majority of $v \in P_n$. Also, maybe the binary search can be dropped and use a method to round the lengths in $O(1)$ time.

2.3 kD-SS approx

We will generalize the same algorithmic idea for arbitrary dimension k. The input set of vectors is again $S = \{v_1, v_2, \ldots v_n\}$ but $v_i \in \mathbb{Z}^k$. It is easier to consider the polar, or spherical, coordinates for each vector: every vector is $v = (r, \phi_1, \ldots, \phi_{k-1})$ where $r = |u|$ is the magnitude and ϕ_i the angle with the i-th axis.

The algorithm is almost the same. For $u \in E_i$ consider all vectors $u' \in E_i$ such that $|u| \leq |u'| \leq (1+\delta)|u|$. Let $\theta_i(u)$ be the angle ϕ_i of u. Next check all angles, if $\forall i : \theta_i(u) - \delta \leq \theta_i(u') \leq \theta_i(u) + \delta$, remove u'. This process leads to a similar result as in eq. (2.3) although, now α depends on the dimension, $\alpha = \Theta(\sqrt{k})$ and

$$\forall u \in E_i \, \exists w \in L_i : u = w + r_w, \; |r_w| \leq \sqrt{k}\delta|w|$$

This implies that $|w| \leq |u| \leq (1+\sqrt{k}\delta)|w|$.

The space is first divided by the lengths of the vectors creating $O(n \log M_n/\epsilon)$ concentric spheres with center the origin. Then, every separation based on the angles creates n/ϵ new cells and for all $k-1$ angles this is $O((n/\epsilon)^{k-1})$. In total, the size of the array E_i can have $O(n^k \epsilon^{-k} \log M_n)$ cells in the worst case. Because this algorithm is quadratic in $|E_i|$ it becomes very slow quickly with a total running time $O(n^{2k+1}\epsilon^{-2k} \log M_n)$

If we separate the space in a grid with every cell having side length $d = \epsilon M_n/n$ for each dimension we create at most $M_n/d = n/\epsilon$ cells. We have k dimensions therefore the size of the array is $O((n/\epsilon)^k)$ and the total running time is $O(n^{k+1}\epsilon^{-k})$. The maximum length inside a cell is its diagonal with length $d\sqrt{k}$. With this setting every vector in E_i is approximated by some vector in L_i that is at most $\epsilon\sqrt{k}M_n/n$ far and every vector in P_n can be approximated be some vector in L_n that is at most $\epsilon\sqrt{k}M_n$ far. If we set $d = \epsilon M_n/(n\sqrt{d})$ the running time becomes $O(n^{k+1}\sqrt{k}^k\epsilon^{-k})$ but the approximation ration drops to ϵM_n.

Corollary 19. *For the problem kD-SS-opt the grid algorithm returns a solution t' such that, either $dist(t,t') \leq OPT + \epsilon\sqrt{k}M_n$ in time $O(n^{k+1}\epsilon^{-k})$ or $dist(t,t') \leq OPT + \epsilon M_n$ in time $O(n^{k+1}\epsilon^{-k}k^{k/2})$.*

Chapter 3

Minkowski Decomposition

MinkDecomp has its fair share of attention. One application is in the factorization of bivariate polynomials through their Newton polygons. As noticed by Ostrowski in 1921, if a polynomial factors, then its Newton polygon has a Minkowski decomposition. An algorithm for polynomial irreducibility testing using MinkDecomp is presented in [KM08] motivated by previous similar work in [Gao01]. They present a criterion for MinkDecomp that reduces the decision problem into a question in linear programming. Continuing the work in [ET06, sections 4,5], we propose a polynomial time algorithm that solves MinkDecomp approximately using a solver for $2D$-SS. Here, we are interested in finding approximate solutions: polygons whose Minkowski Sum is almost the original polygon.

Knowing if and how a polygon decomposes can be useful in other fields. Namely implicitization where we have the parametric representation of an object and we want to find its algebraic, or implicit, representation. Another topics can be tropical geometry where one operation can be the Minkowski sum of polygons and its reverse is Minkowski decomposition.

The decision version of problem 2 (whether a polygons admits a Minkowski decomposition) is proven NP-complete even for dimension 2 in [GL01] using a reduction to the Partition problem. Also in [GL01] the authors propose an pseudo-polynomial time algorithm to solve the problem in time $O((nD)^3)$. In [ET06] a different reduction from Subset-Sum is given and an improved algorithm is presented with running time $O((nD)^2)$. This last bound is tight as can be seen in [ET06, sec. 2]. Here DE is the diameter of the polygon.

We will briefly present the NPcompleteness proof and in the next sections we will describe an approximation algorithm for MinkDecomp. The algorithm takes as input polygon Q, transforms it into an instance $\{S, t\}$ of $2D$-SS-approx and calls algorithm 1 for $2D$-SS-approx. Then it takes the output and converts it into an approximate solution to MinkDecomp.

An approach, apart from dynamic programming, to overcome the difficulty of NPhard problems is to device approximation algorithms that return a near optimal solution. For this task we will define the approximation version of MinkDecomp.

Problem 20. MinkDecomp-μ-approx

Input: A lattice polygon P, a parameter $0 < \epsilon < 1$ and a function μ.

Output: Lattice polygons A, B such that $0 \leq \mu(A+B) - \mu(P) < \epsilon \cdot \phi(D)$, where D is the diameter of P and $\phi()$ a polynomial. We call such an output an $\epsilon \cdot \phi(D)$-solution.

For μ we will consider the functions $vol(P)$: the volume, $per(P)$: the perimeter and $latt\_points(P)$: the internal lattice points of P ($\#(P \cap \mathbb{Z})$, but also we consider

$d_H(P, A + B)$: the Hausdorff distance between P and $A + B$. Other functions suitable for comparison of shapes is the Fréchet distance between curves and the turning function of a polygon but we do not have results for these.

3.1 MinkDecomp is NP complete

Recall the definition of the problem and restrict it to polygons.

Problem. $2D$**-MinkDecomp**
Given a lattice convex polygon $P \subset \mathbb{Z}^2$, decide if it is decomposable, that is, if there are nontrivial lattice polygons A and B such that $A + B = Q$, where $+$ denotes Minkowski addition.

Let $P_1 <_p P_2$ denote a polynomial time reduction from problem P_1 to P_2.

Theorem 21. *Partition* $<_p$ *2D-MinkDecomp* $<_p$ *2D-SS*

Proof. Partition $<_p$ $2D$-MinkDecomp
The input is set $S = \{a_1, \ldots, a_n\}$ of integers that $\sum S = t$ and we want a subset of S that sums to $t/2$. Sort the numbers in increasing order and create the vectors $v_i = (a_i, 1), \forall a_i \in A$. Also, take n times the vector $(0, -1)$ and two times $(-t/2, 0)$. Notice that the sum of all these vectors equals $(0, 0)$ and they form a polygon (see figure 3.1). Suppose there is $S' \subset S$ such that $|S'| = k$ and $\sum S' = t/2$. Take the corresponding vectors $v_i, \forall a_i \in S'$, k vertical vectors that sum to $(0, -k)$ and one horizontal vector that sum to $(-t/2, 0)$. These sum to $(0, 0)$ and form a polygon that is a summand of P.

On the other direction, if there is a summand A of P then $(-t/2, 0)$ must be in its edge sequence. Also, $\sum s(A) = (0, 0)$ and in order to achieve this there must be vectors v_i that sum to $(t/2, k)$ and $k(0, -1)$ vectors. Note that the a_i for every v_i add up to $t/2$ which means that the partition problem has a solution.

$2D$-MinkDecomp $<_p$ $2D$-SS
Given a polygon P get its edge sequence $s(P)$ and the set S as described. The set S and target $t = (0, 0)$ are the inputs of $2D$-SS. If there are polygons A, B such that $A + B = P$ then $\sum s(A) = (0, 0)$ and also $\sum s(B) = (0, 0)$ so, there must be a subset in S that corresponds to $s(A)$ that sums to zero since $s(A) \subset S$. On the other hand, if $\exists S' \subset S$ such that $\sum S' = (0, 0)$ we could take $s(A) = S'$ and form polygons A and $s(B) = S \setminus S'$ for B that yield $A + B = P$.

□

Since Partition is NP-complete both MinkDecomp and $2D$-SS are NP-hard. Given a solution to either of them is easy to check its correctness (Minkowski sum can be done in linear time) making them both NP-complete.

3.2 Solve MinkDecomp Using 2D-SS

Let Q be the input polygon to MinkDecomp. First we create the vector set $s(Q)$ by subtracting successive vertices of Q (in clockwise order): $s(Q) = \{v_0 - v_1, v_1 - v_2, \ldots, v_n - v_0\}$. Each vector in $s(Q)$ is called an **edge vector** and $s(Q)$ the **edge**

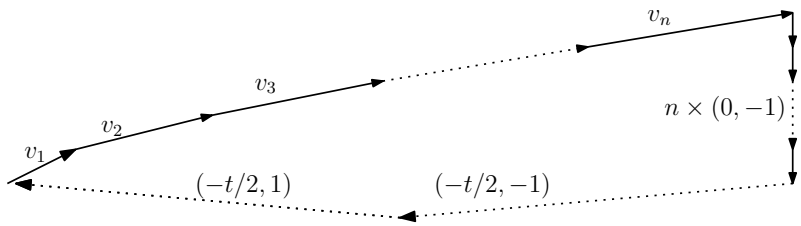

FIGURE 3.1: The polygon from the reduction of $1D$-SS to $2D$-MinkDecomp.

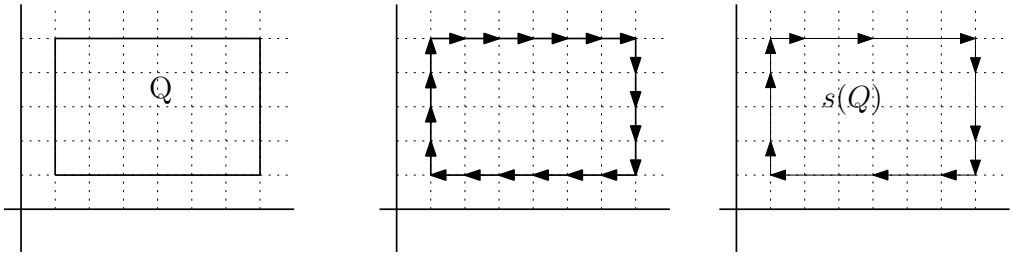

FIGURE 3.2: An example where the polygon has 4 vertices but, with the naive approach, its edge sequence 28 vectors. In a worst case, if we are not careful, $s(Q)$ could have exponential size compared to the size of Q. The actual $s(Q)$ now has only 12 vectors instead of 28.

sequence of Q. For each edge vector in $s(Q)$ we calculate its *primitive vector.* It is easy to see that if Q is decomposable into A and B such that $Q = A + B$, then $s(Q) = s(A) \cup s(B)$. In words, the edges of Q are exactly the edges of the two polygons figure 3.3.

Algorithm 3: approx-MinkDecomp

input : Q, ϵ
output: Q'

S = primitive_edge_sequence(Q)
s(A) = approx-2D-SS (S,(0,0)) //get the edge sequence for the two summands
s(B) = S\A
A=get-points(s(A)) //returns a polygon fom an input edge sequence
B=get-points(s(B))
return $Q' = A + B$

Definition 22. Let $v = (a, b)$ be a vector and $d = gcd(a, b)$. The **primitive vector** of v is $e = (a/d, b/d)$.

We must construct the vector set S that will be the input to to $2D$-SS solver. Get an edge vector $(x, y) \in s(Q)$ and calculate its primitive vector $e = (x/d, y/d)$, where $d = gcd(x, y)$. We could create the set S by adding in it d times the vector e for every $v \in s(Q)$ but this may create a set S that has exponential size to the number of edges of the polygon. For example the edge sequence of a rectangle can be very large if we do it this way, see figure 3.2.

Instead, we computer the scalars $d_1, \ldots, d_k$ by the following formulas

$$d_i = 2^i,\ i = 0, \ldots, \lfloor \log_2 d/2 \rfloor \text{ and } d_k = d - \sum_{i=1}^{\lfloor \log_2 d/2 \rfloor} d_i$$

We create the set S from the vectors $d_i e$ and repeat the procedure for all vectors $v \in s(Q)$. Notice that $\sum_1^k d_i = d$, so this edge sequence also sums to $(0, 0)$. (For example, if $d = 110$ the scalars d_i will be $1, 2, 4, 8, 16, 32, 47$ and every number between 1 and 110 can be expressed as a combination of these numbers.) Using this construction, the vectors added are at most $\log d$ for every $v \in s(Q)$, keeping the size of S polynomial with respect to the number of edges of Q. (Given a square Q with sides parallel to the axes, $|s(Q)| = n \log D$ since the length of each side is almost D.) The primitive edge sequence uniquely identifies the polygon up to translation determined by v_0. This is a standard procedure as in [GL01; ET06].

A simple heuristic can be introduced here. The idea is to increase the vectors in $s(Q)$ because this can give as more possible vector sums and thus more possible solutions. Suppose that for an edge vector $u \in s(Q)$ its coordinates are co prime. This edge cannot be separated to a smaller primitive vector. It has to be used in a sum undivided. If we perturb its coordinates a bit, only by adding or subtracting 1, we alter the vector slightly but we can now divide the edge at least one time but hopefully more. For example consider the vector $u = (31, 21)$ with length $|u| = 37.44$ that has no lattice point, its indivisible $gcd(u) = 1$. If we change the coordinates a bit, we can take the vector $u' = (33, 21)$ instead that can be divided 3 times and has length $|u'| = 39.11$ or the vector $u'' = (30, 20)$ that can be divided 10 times and has length $|u''| = 36.05$. Since an approximate solution already contributes at most ϵM_n, adding such a small factor is negligible but can greatly increase the possibilities of finding a better solution. Consider for example the case where no edge contains lattice points and $|s(Q)| = m$. Using this heuristic each edge will be divided at least once doubling the size of the new $s'(Q)$ and greatly helping towards a possible solution. We cannot prove an actual improvement thus we regard this technique a heuristic.

Now we can give this edge sequence of the polygon to the algorithm for $2D$-SS-approx with target $(0, 0)$. If Q is decomposable in A and B such that $A + B = Q$ then $\sum s(A) = 0$ since A is a closed polygon (the same for B) and $s(A) \subset s(Q)$. Thus, there exists a subset of $s(Q)$ that sums to $(0, 0)$. See also theorem 21 for the reduction to $2D$-SS. The main defect in this approach is that the approximation algorithm returns a sequence of vectors $S' \subset S$ that sum close to the origin but probably not exactly to $(0, 0)$. This means that the corresponding edge sequence does not sum to $(0, 0)$ and does not form a closed polygon. To overcome this, we add the missing vector to force the sequence to sum to zero (figure 3.4).

If $s(A)$ sums to a point (a, b), by adding vector $v = (-a, -b)$ to $s(A)$ the edge sequence $s(A) \cup \{v\}$ now sums to $(0, 0)$. If we rearrange the vectors by their angles, they form a closed, convex polygon that is summand A. We do the same for the sequence $s(B)$. The vector added in $s(B)$ is $-v = (a, b)$ and this sequence (rearranged) also forms a closed, convex polygon. We name $s(A') = s(A) \cup \{v\}$, $s(B') = s(B) \cup \{v\}$ and take their Minkowski Sum $Q' = A' + B'$, where A' and B' are the convex polygons formed by $s(A')$ and $s(B')$. We measure how close Q' is

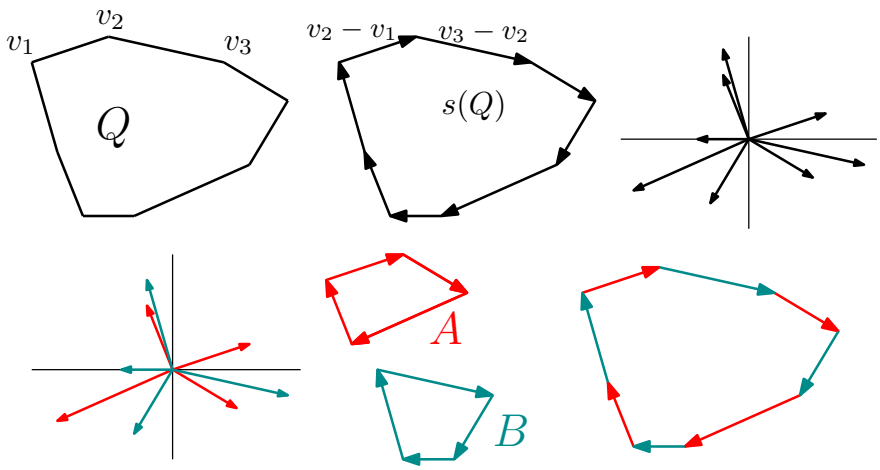

FIGURE 3.3: How from a polygon Q we get its edge sequence $s(Q)$ and transform it to an instance of $2D$-SS. See that $s(A) + s(B) = s(Q)$.

to the input Q. Let D be the diameter of Q, the maximum distance between two vertices of Q.

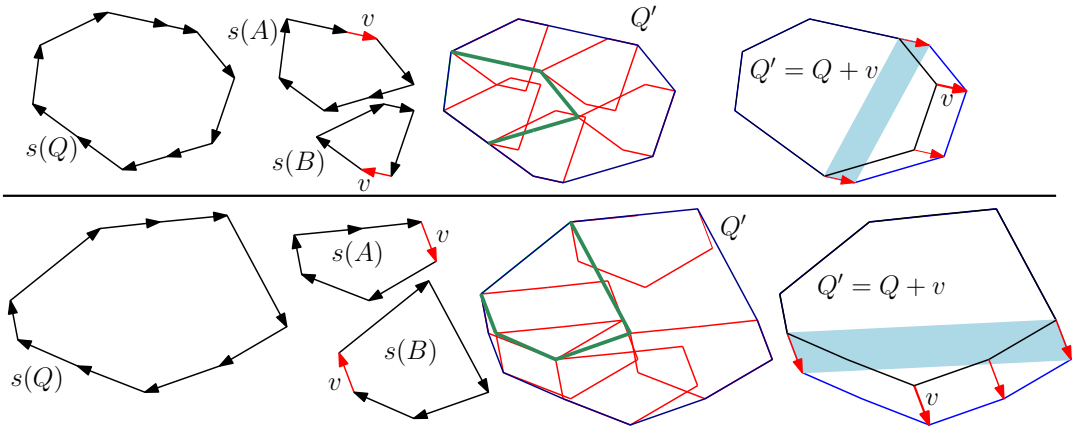

FIGURE 3.4: Two examples for two polygons Q. Their summands are shown and the red vector v is the new vector added to fill the gap. At the end, the new polygon Q' is Minkowski Sum of the two summands.

3.3 Results for approximating MinkDecomp

The approximation algorithms gives some results but how the returned polygon compares to the input? Actually the subject of shape comparison and measuring similarity is not as trivial as it may seem at first [Vel01; VH01]. And probably, comparing one measure alone is not enough. For example a circle and a rectangle can

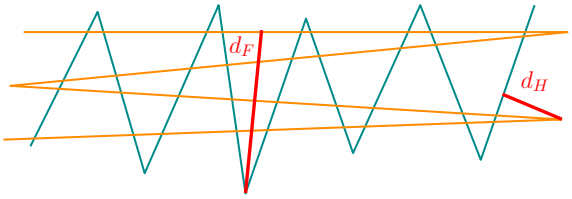

FIGURE 3.5: The Hausdorff d_H and Fréchet d_F distance between two curves.

have the same volume, or perimeter, but are very different. At least how humans perceive the difference between a circle and a rectangle.

Another way is to measure their distance. Two important functions for distance is the Hausdorff and the Fréchet distance, d_H and d_F respectively. Computing the Hausdorff between two point sets with n points in $\mathbb{R}^2$ can be done in $O(n \log n)$ time using Voronoi diagrams. For the Fréchet distance thing seem more complicated and it has been well studied. Several algorithm exist, both exact and approximate, with polynomial running time. One with subquadratic algorithm recently appeared in [Aga+12]. The seminal paper is from Alt and Godau [AG95] and other interesting recent papers are [Aga+12; Bri14]. In our case the two distances are almost equal but further work can be done here for the bound in the Fréchet distance.

Lemma 23. *Let Q be the input polygon and Q' be the polygon as discussed above. vol stands for volume, per for perimeter, $latt\_p$ are the interior lattice points of a polygon and d_H the Hausdorff distance. We deduce that*

1. $vol(Q') - vol(Q) \leq \epsilon D^2$
2. $per(Q') - per(Q) \leq 2\epsilon D$
3. $latt\_p(Q') - latt\_p(Q) \leq \epsilon D^2$
4. $d_H(Q, Q') \leq \epsilon/2D$

Proof. We observe that

$$s(Q') = s(A') \cup s(B') = s(A) \cup s(B) \cup \{v\} \cup \{-v\} \implies$$
$$s(Q') = s(Q) \cup \{v\} \cup \{-v\}.$$

This equals adding to Q a single segment s of length $|s| = |v|$ and $Q' = Q + s$. The length of vector v we add to close the gap, is the key factor to bound polygon Q'. From the guarantee of the $2D$-SS-approx algorithm we know that $s(A)$ (and respectively $s(B)$) sum to a vector with length at most $\epsilon \max\{L_n\}$. This is vector v and thus $|v| \leq \epsilon \max\{L_n\}$. Since $\max\{L_n\} \leq D$, we get $|s| = |v| \leq \epsilon D$.

From above we easily see that:

1. $per(Q) = \sum_{v \in s(Q)} |v|$, it follows $per(Q') = per(Q) + 2|v| \leq per(Q) + 2\epsilon D$.
2. $vol(Q') \leq vol(Q) + sD \leq vol(Q) + \epsilon D^2$
3. By Pick's theorem, $vol(Q) = i(Q) + b(Q)/2 - 1 \implies i(Q) = vol(Q) - b(Q)/2 + 1$. Note that $b(Q) = \sum_{\forall v \in s(Q)} d_v$ where $v = (x, y) \in s(Q)$ and $d_v = \gcd(x, y)$ as is Definition 22. Now, $i(Q') = i(Q) + i(sD)$ since sD is the maximum volume added and $i(sD) \leq sD - b(sD)/2 + 1 \leq sD - 1 \leq \epsilon D^2$. Thus, $i(Q') \leq i(Q) + \epsilon D^2$

4. The two polygons are as in figure 3.4, their Hausdorff distance is $|v|$. If we "slide" Q by $|v|/2$ units in the direction of v, $d_H(Q, Q') = s/2 \implies d_H(Q, Q') \leq \epsilon/2D$

□

A bad example can be seen in figure 3.6 where the vector we must add is (almost) perpendicular to D maximizing the extra volume and internal lattice points. Lemma 23 leads to following conclusion:

Corollary 24. *The proposed algorithm provides a $2\epsilon D$-solution for MinkDecomp-per-approx, a ϵD^2-solution for MinkDecomp-vol-approx and MinkDecomp-latt_p-approx and a $\epsilon/2D$-solution for MinkDecomp-d_H-approx.*

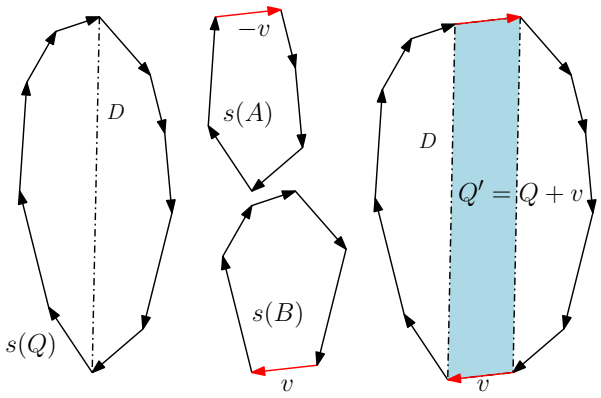

FIGURE 3.6: A worst case example where the vector v is (almost) perpendicular to the diameter D maximizing the extra volume added. More, D and v have no lattice points thus the interior points added are also maximum (D is not vertical).

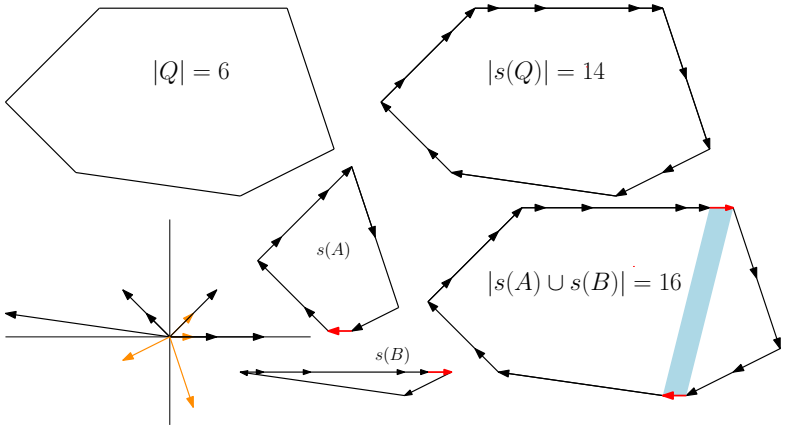

FIGURE 3.7: Another example how from a polygon Q we get its edge sequence, we find a subset of $s(Q)$ that sums close to $(0, 0)$ and form the two summands that give an approximation to out input.

Chapter 4

Conclusion and open problems

To summarize the present thesis, we examined a new approach to solve the NP-complete problem MinkDecomp by approximating it. MinkDecomp is connected to the Subset Sum problem and this lead us to define a generalization of Subset Sum to arbitrary dimension k where instead of numbers we have k-dimensional vectors. We were able to provide solutions in polynomial time that are ϵM far from the optimum for kD-SS. Using this algorithm, and the reduction from Minkdecomp to kD-SS, we can provide solutions to MinkDecomp-approx. We measured these solutions using different functions like volume,internal lattice points or Hausdorff distance.

The final goal is to see how, from an approximate solution to MinkDecomp we can get solutions to questions related to polynomials. It turns out that is not straightforward; if it can be done at all. We also came up with questions that are left unanswered at the moment. All these consist the future work that has to be done in this subject. As it turned out we end up with more questions than answers and certainly more than when we started.

4.1 kD-SS

Results and answers here concern theoretical and practical aspects. The algorithms proposed can return solutions that their error depend on the longest vector in time polynomial to the input and the error ratio ϵ. Since the problem is defined here, we do not have prior results to compare to ours. Can the algorithms and approximation ratio be improved in any way? For low constant dimensions can we do better? If we restrict to specific cases, say for example when all vectors are positive, can we provide better results? Through experimental tests, it appears that the algorithm is way faster and more precise in this case. We also believe that with a probabilistic analysis better bound can be proven for the average case.

On the theoretical side, as a generalization of the classic $1D$-SS the problem is easily NP-complete. Also, for general dimension k, it cannot be approximated within a constant factor. It is connected, as a special case, to the well-studied CVP that is even harder to approximate. Is the same true for kD-SS? can we prove the same inapproximability results as for CVP or is it easier. If we consider constant dimensions, even for $k = 2$ or 3, can the problem be approximated better or is still hard? Answers here will adequately characterize the problem when it comes to approximate solutions. Apart from approximation, where the problem stands concerning other complexity classes.

A seemingly similar know problem is Multidimensional Knapsack (MK). The $1D$-SS is a straight special case of Knapsack when the weight of each object equals its value. For their multidimensional versions it depends on the definition. For the classic 1-dimensional case, their optimization version are defined as maximization problems. For Knapsack, we want to maximize the value of all the object without exceeding the knapsack's capacity. In Subset Sum we want a subset of number that have maximum sum but less than target t. We defined the kD-SS as an minimization problem: we are looking for a subset that sums to t' and minimizes the distance from target t. The difference in the definitions makes kD-SS a harder problem to approximate.

We can give a similar definition for the $1D$ case: a subset that sums to t' and minimizes the quantity $t - t'$. These two versions are the same in the sense that the optimal solution for the maximization problem is also optimal for the minimization. In two dimensions (and higher) this is not correct.

4.2 MinkDecomp

The algorithm for approximating $2D$-SS directly applies here. Through the transformation of a polygon Q to its vector sequence we can solve MinkDecomp by first solving $2D$-SS. The resulting polygon Q' is bounded in many ways compared to the original as already seen in section 3.2. We believe that for the Fréchet distance an actual bound can be proved. Maybe the same can be done for other functions used in polygon comparison like the turning function.

In section 3.2 we introduce a heuristic where each vertex is slightly perturbed in order to be able to divide in smaller segments. Can this idea give us proven better results? If we know that every edge of the input polygon can be divided at least once (or maybe more), is the problem easier? (This is equivalent to also allow multiplication with $1/2$.) This can be stated as follows. Given a polygon Q that we also know that every edge of Q can be divided exactly d times, decide if it is decomposable. This problem is reduced to the problem of finding a subset S' in the set R where now, R is not $s(Q)$ but is the set of all primitive vectors of Q, such that,

$$\sum_{v \in S'} av = (0, 0), \ a \in \{0, 1, \ldots, d\}$$

If we know that the vectors in $s(Q)$ can be divided at least d times, we get the set of its primitive vectors and allow multiplication with a scalar up to d. In some sense, this approaches CVP and probably is as difficult. Maybe dynamic programming can be more effective now. This proposes an algorithm that first slightly perturbs the vertices of the polygon in a suitable manner and then uses dynamic programming to solve exactly this new perturbed instance. This also raises the question if this perturbation of all vertices can be done efficiently because changing one vertex affects two vectors.

Since MinkDecomp is a natural geometric problem other, completely new approaches can be used that do not have a connection with Subset Sum. On a theoretical view the problem is not given much consideration apart from its NP-completeness. A related problem is the k-summand where we search for polygons

with k edges [ET06]. What is the difficulty of this parametric version of the problem? Classifying this in the parameterized complexity hierarchy might give further insight to the problem. Equivalently, how difficult is the problem with respect to approximation?

Apart from polygons we can generalize the problem to point set. For example, what is the difficulty of the variation where we do not have polygons to decompose but point sets? A polygon is a point set with a specific property thus the problem defined for point sets is more general. An approach in this variation maybe enable us to work with the support $Supp(f)$ of the polynomial f instead of its Newton polytope. The support holds more information since interior points do not disappear and maybe can give different kind of results. A related question was proposed by Sturmfels: given the support of a polynomial to compute the support of all possible factorizations. In [GL01] the authors give a partial solution using the Newton polytope. Perhaps this can be approached by similar methods but on the support instead of the Newton polytope.

4.2.1 From polygons back to polynomials

It is easy and quick to retrieve a polygon (the Newton polygon or polytope for higher dimensions) given a polynomial. Every polynomial has a unique support and thus a unique Newton polytope. The reverse is not so trivial. Given a polygon P it can be the Newton polygon of many polynomials. Name this family $\mathcal{F}_P$, ie. the set of polynomials that their Newton polygon equals P,

$$\mathcal{F}_P = \{f \in \mathbb{F}[x, y] \mid NP(f) = P\}$$

Because a polygon does not come from a unique polynomial, retrieving a suitable polynomial from $\mathcal{F}_P$ is not easy.

Applying the algorithm for MinkDecomp-approx in the Newton polygon Q of a bivariate polynomial f will return two other polygons A and B such that $A + B = Q'$, thus Q' is decomposable. This does not give any information about the original Q; it may be decomposable or indecomposable, we do not know. Thus, this approach cannot give answers concerning the irreducibility testing of a polynomial because it completely alters the Newton polytope.

The news are not so encouraging for approximate factoring either. Ostrowski's theorem works on one direction, if the Newton polytope of a polynomial decomposes does not mean that the polynomial can be factored. Nonetheless, we can make some observations on this matter. Besides the two polygons A and B the algorithm finds a single vector t' (with $|t'| \leq \epsilon D$) such that $Q' = Q + t'$ and

$$Q' = A + B = Q + t'$$

Because t' is just a vector, $NP^{-1}(t') = x^a y^b + c$ where a, b are integers and c is a constant and also $a + b \leq \epsilon D$ where D is less than the total degree of f. Let f_P be a polynomial such that $f_P = NP(P)$ for some polygon P. Given f_Q, we can state

the question: are there polynomials $f_A, f_B, x^a y^b + c$ such that,

$$f_Q = \frac{f_A f_B}{x^a y^b + c} = \frac{f_{Q'}}{x^a y^b + c}$$

knowing that $Q' = A + B = Q + t'$. And if those polynomials exist how they can be found?

Towards this goal are the results of Salem, Gao, and Lauder [SGL04] and the PhD thesis of A. Salem [Sal04, chap. 6]. There the authors consider the problem of factorizing a polynomial f or deciding irreducibility if they also know polygons P_f, A, B such that $P_f = A + B$. They propose an algorithm for that task under minor conditions that runs in time cubic to the $latt\_p(P_f)$. Note that this is not polynomial to the input since the number of integer lattice points can be exponentially many compared to the number of vertices of P_f. Combining the algorithms here with their results is a future goal that may offer an approximation algorithm for factoring.

4.3 Implementation and experiments

We implement both Algorithms 1 and 3 in Python3. The code can be accessed through Github [1] and is roughly 750 lines long. We provide methods for either $2D$-SS-approx or MinkDecomp-μ-approx. To test algorithm 1 we created vectors v_i at random with $|v_i| \leq 5000$. For algorithm 3 we create random points and take their convex hull to form input polygon Q. All tests were executed in an Intel Core i5-2320 @ 3.00 GHz with 8Gb RAM, 64-bit Ubuntu GNU/Linux. Results for algorithm 1 are shown in Figure 4.1 and for algorithm 3 in table 4.1. It is clear in figure 4.1 that our results stay way below the expected time and behave analogously. In table 4.1 the results obtained are much better than the proven bounds and in most cases volume and perimeter are almost the same and the polygons differ slightly.

#vertices	#examples	vol(Q)/vol(Q')	per(Q)-per(Q')	Hausdorff	ϵ	time(secs)
3—10	51	0,93	18,55	3,32	0,18	4,1
11—16	45	0,977	3,43	1,81	0,33	126,4
17—25	54	0,994	1,12	1,25	0,38	377,5

TABLE 4.1: Input polygon Q, output Q' ($per(Q) > 1000$). Gather examples by the number of their vertices. We measure volume, perimeter and Hausdorff distance and present their mean values.

[1] https://github.com/tzovas/Approximation-Subset-Sum-and-Minkowski-Decomposition

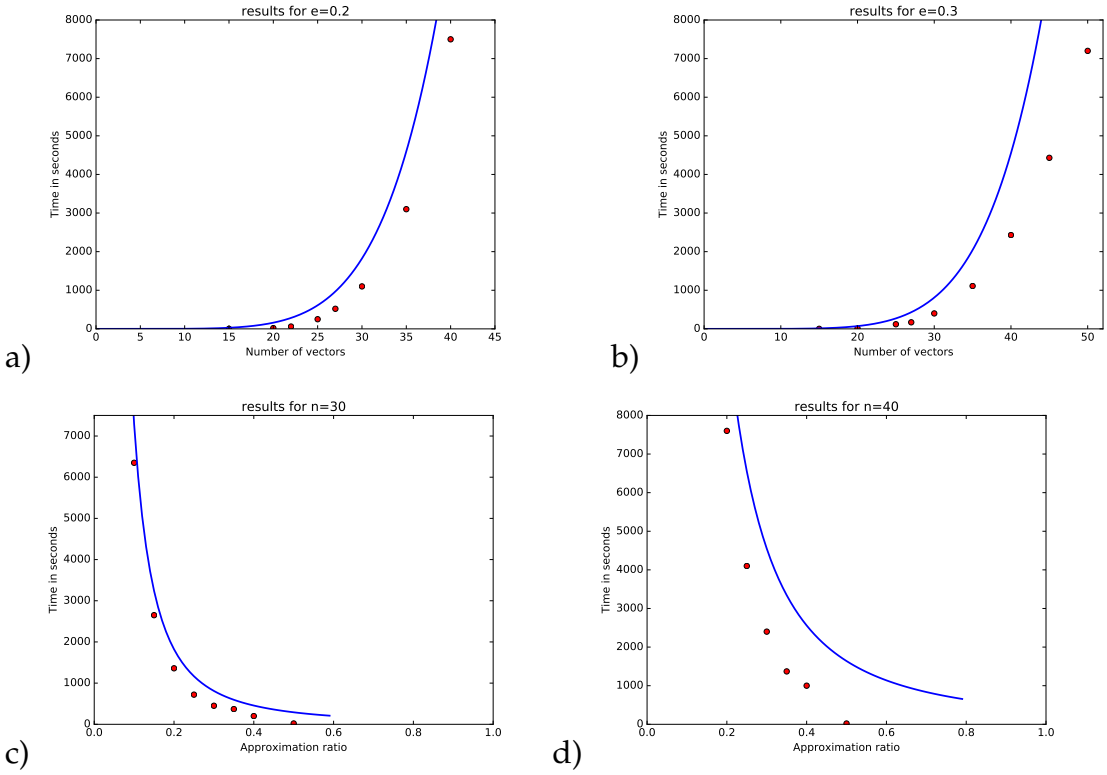

FIGURE 4.1: Experimental results for $2D$-SS-approx: a)$\epsilon = 0.2$ and b)ϵ=0.30, c)$n = 30$ and d)$n = 40$. The blue line is the expected time, the red dots our experiments.

Bibliography

Agarwal, Pankaj K. et al. (2012). "Computing the Discrete Fréchet Distance in Subquadratic Time". In: *CoRR* abs/1204.5333. URL: http://arxiv.org/abs/1204.5333.

ALT, HELMUT and MICHAEL GODAU (1995). "COMPUTING THE FRECHET DISTANCE BETWEEN TWO POLYGONAL CURVES". In: *International Journal of Computational Geometry and Applications* 05.01n02, pp. 75–91. DOI: 10.1142/S0218195995000064.

Arkin, E. M. et al. (1991). "An efficiently computable metric for comparing polygonal shapes". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.3, pp. 209–216. ISSN: 0162-8828. DOI: 10.1109/34.75509.

Arora, Sanjeev et al. (1997). "The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations". In: *Journal of Computer and System Sciences* 54.2, pp. 317 –331. ISSN: 0022-0000. DOI: http://dx.doi.org/10.1006/jcss.1997.1472. URL: http://www.sciencedirect.com/science/article/pii/S0022000097914720.

Belabas, K. et al. (2004). "Factoring polynomials over global fields". In: *ArXiv Mathematics e-prints*. eprint: math/0409510.

Belabas, Karim (2004). "A relative van Hoeij algorithm over number fields". In: *Journal of Symbolic Computation* 37.5, pp. 641 –668. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/j.jsc.2003.09.003.

Bellare, M. et al. (1993). "Efficient Probabilistically Checkable Proofs and Applications to Approximations". In: *Proceedings of the Twenty-fifth Annual ACM*. STOC '93. San Diego, California, USA: ACM, pp. 294–304. ISBN: 0-89791-591-7. DOI: 10.1145/167088.167174. URL: http://doi.acm.org/10.1145/167088.167174.

Berlekamp, E. R. (1967). "Factoring Polynomials Over Finite Fields". In: *Bell System Technical Journal* 46.8, pp. 1853–1859. ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1967.tb03174.x. URL: http://dx.doi.org/10.1002/j.1538-7305.1967.tb03174.x.

Bostan, A. et al. (2004). "Complexity Issues in Bivariate Polynomial Factorization". In: *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation*. ISSAC '04. New York, NY, USA: ACM, pp. 42–49. ISBN: 1-58113-827-X. DOI: 10.1145/1005285.1005294. URL: http://doi.acm.org/10.1145/1005285.1005294.

Bringmann, Karl (2014). "Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails". In: *CoRR* abs/1404.1448. URL: http://arxiv.org/abs/1404.1448.

Brown, Ron (2008). *Roots of Generalized Schonemann Polynomials in Helselian Extension Fields*.

Caprara, Alberto, Hans Kellerer, and Ulrich Pferschy (1998). "The Multiple Subset Sum Problem". In: *SIAM JOURNAL OF OPTIMIZATION* 11, pp. 308–319.

Caprara, Alberto et al. (2000). "Approximation algorithms for knapsack problems with cardinality constraints". In: *European Journal of Operational Research* 123.2, pp. 333 –345. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/S0377-2217(99)00261-1. URL: http://www.sciencedirect.com/science/article/pii/S0377221799002611.

Coppersmith, Don and Shmuel Winograd (1990). "Computational algebraic complexity editorial Matrix multiplication via arithmetic progressions". In: *Journal of Symbolic Computation* 9.3, pp. 251 –280. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/S0747-7171(08)80013-2.

Cormen, Thomas H. et al. (2009). *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press. ISBN: 0262033844, 9780262033848.

Cox, David A. (2011). "Why Eisenstein Proved the Eisenstein Criterion and Why Schonemann Discovered It First". In: *The American Mathematical Monthly* 118.1, pp. 3–21. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/10.4169/amer.math.monthly.118.01.003.

David G. Cantor, Hans Zassenhaus (1981). "A New Algorithm for Factoring Polynomials Over Finite Fields". In: *Mathematics of Computation* 36.154, pp. 587–592. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2007663.

Dinur, I. et al. (2003). "Approximating CVP to Within Almost-Polynomial Factors is NP-Hard". English. In: *Combinatorica* 23.2, pp. 205–243. ISSN: 0209-9683. DOI: 10.1007/s00493-003-0019-y. URL: http://dx.doi.org/10.1007/s00493-003-0019-y.

Emiris, Ioannis. and Elias. Tsigaridas (2006). "Minkowski decomposition of convex lattice polygons". English. In: *Algebraic Geometry and Geometric Modeling*. Ed. by Mohamed Elkadi, Bernard Mourrain, and Ragni Piene. Mathematics and Visualization. Springer Berlin Heidelberg, pp. 217–236. ISBN: 978-3-540-33274-9. DOI: 10.1007/978-3-540-33275-6_14.

Emiris, Ioannis Z., Christos Konaxis, and Zafeirakis Zafeirakopoulos (2015). "Minkowski Decomposition and Geometric Predicates in Sparse Implicitization". In: *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation*. ISSAC '15. Bath, United Kingdom: ACM, pp. 157–164. ISBN: 978-1-4503-3435-8. DOI: 10.1145/2755996.2756661. URL: http://doi.acm.org/10.1145/2755996.2756661.

Frieze, A.M. and M.R.B. Clarke (1984). "Approximation algorithms for the m-dimensional 0-1 knapsack problem: Worst-case and probabilistic analyses". In: *European Journal of Operational Research* 15.1, pp. 100 –109. ISSN: 0377-2217. DOI: http://dx.doi.org/10.1016/0377-2217(84)90053-5. URL: http://www.sciencedirect.com/science/article/pii/0377221784900535.

Gall, François Le (2014). "Powers of Tensors and Fast Matrix Multiplication". In: *CoRR* abs/1401.7714. URL: http://arxiv.org/abs/1401.7714.

Gao, S. and A. G B Lauder (2001). "Decomposition of polytopes and polynomials". In: *Discrete and Computanional Geometry* 26.1, pp. 89–104. ISSN: 0179-5376.

Gao, Shuhong (2001). "Absolute Irreducibility of Polynomials via Newton Polytopes". In: *Journal of Algebra* 237.2, pp. 501 –520. ISSN: 0021-8693. DOI: http://dx.doi.org/10.1006/jabr.2000.8586.

— (2003). "Factoring multivariate polynomials via partial differential equations". In: *Math. Comp. 72 (2003), 801-822*. DOI: http://dx.doi.org/10.1090/S0025-5718-02-01428-X.

Gao, Shuhong and Joachim von zur Gathen (1994). "Berlekamp's and Niederreiter's Polynomial Factorization Algorithms". In: *Finite Fields: Theory, Applications and Algorithms*.

Gao, Shuhong, Erich Kaltofen, and Alan G.B. Lauder (2004). "Deterministic distinct-degree factorization of polynomials over finite fields". In: *Journal of Symbolic Computation* 38.6, pp. 1461 –1470. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/j.jsc.2004.05.004. URL: http://www.sciencedirect.com/science/article/pii/S0747717104000732.

Gao, Shuhong and Alan G. B. Lauder (2004). *Fast Absolute Irreducibility Testing via Newton Polytopes*. Tech. rep.

Gao, Shuhong and Daniel Panario (1997). "Foundations of Computational Mathematics: Selected Papers of a Conference Held at Rio de Janeiro, January 1997". In: ed. by Felipe Cucker and Michael Shub. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Tests and Constructions of Irreducible Polynomials over Finite Fields, pp. 346–361. ISBN: 978-3-642-60539-0. DOI: 10.1007/978-3-642-60539-0_27. URL: http://dx.doi.org/10.1007/978-3-642-60539-0_27.

Gathen, Joachim von zur (1985). "Irreducibility of multivariate polynomials". In: *Journal of Computer and System Sciences* 31.2, pp. 225 –264. ISSN: 0022-0000. DOI: http://dx.doi.org/10.1016/0022-0000(85)90043-1. URL: http://www.sciencedirect.com/science/article/pii/0022000085900431.

Gathen, Joachim von zur and Victor Shoup (1992). "Computing Frobenius maps and factoring polynomials". In: *computational complexity* 2.3, pp. 187–224. ISSN: 1420-8954. DOI: 10.1007/BF01272074. URL: http://dx.doi.org/10.1007/BF01272074.

Gathen, Joachim Von Zur and Jurgen Gerhard (2003). *Modern Computer Algebra*. 2nd ed. New York, NY, USA: Cambridge University Press. ISBN: 0521826462.

Gathen, Joacin von zur and Erich Kaltofen (1985). "Factorization of Multivariate Polynomials Over Finite Fields". In: *Mathematics of Computation* 45.171, pp. 251–261. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2008063.

Goldman, Ron (2003). "{CHAPTER} 8 - Pyramid Algorithms for Multisided Bezier Patches". In: *Pyramid Algorithms*. Ed. by Ron Goldman. The Morgan Kaufmann Series in Computer Graphics. San Francisco: Morgan Kaufmann, pp. 445 –530. ISBN: 978-1-55860-354-7. DOI: http://dx.doi.org/10.1016/B978-155860354-7/50009-X.

Hoeij, Mark van (2002). "Factoring Polynomials and the Knapsack Problem". In: *Journal of Number Theory* 95.2, pp. 167 –189. ISSN: 0022-314X. DOI: http://dx.doi.org/10.1006/jnth.2001.2763. URL: http://www.sciencedirect.com/science/article/pii/S0022314X01927635.

Ibarra, Oscar H. and Chul E. Kim (1975). "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems". In: *J. ACM* 22.4, pp. 463–468. ISSN: 0004-5411. DOI: 10.1145/321906.321909. URL: http://doi.acm.org/10.1145/321906.321909.

Kaltofen, E. (1982). "Polynomial factorization". In: *Computer Algebra*. Ed. by B. Buchberger, G. Collins, and R. Loos. 2nd ed. SpringerVerl, pp. 95–113.

— (1990). "Polynomial Factorization 1982-1986". In: *Computers in Mathematics*. Ed. by D. V. Chudnovsky and R. D. Jenks. Vol. 125. Lecture Notes in Pure and Applied Mathematics. New York, N. Y.: Marcel Dekker, Inc., pp. 285–309.

— (1992). "Polynomial factorization 1987-1991". In: *Proc. LATIN '92*. Ed. by I. Simon. Vol. 583. SLNCS. SpringerVerl, pp. 294–313.

Kaltofen, E. and V. Shoup (1998). "Subquadratic-time factoring of polynomials over finite fields". In: *MathComp* 67.223, pp. 1179–1197.

Kaltofen, Erich (1985). "Polynomial-Time Reductions from Multivariate to Bi- and Univariate Integral Polynomial Factorization". In: *SIAM J. Comput.* 14.2, pp. 469–489. DOI: 10.1137/0214035. URL: http://dx.doi.org/10.1137/0214035.

— (1987). "Deterministic irreducibility testing of polynomials over large finite fields". In: *Journal of Symbolic Computation* 4.1, pp. 77 –82. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/S0747-7171(87)80055-X. URL: http://www.sciencedirect.com/science/article/pii/S074771718780055X.

— (2003). "Polynomial Factorization: a Success Story". In: *ProcISSAC03*. Abstract for invited talk., pp. 3–4.

Kaltofen, Erich and Austin Lobo (1994). "Factoring High-degree Polynomials by the Black Box Berlekamp Algorithm". In: *Proceedings of the International Symposium on Symbolic and Algebraic Computation*. ISSAC '94. New York, NY, USA: ACM, pp. 90–98. ISBN: 0-89791-638-7. DOI: 10.1145/190347.190371. URL: http://doi.acm.org/10.1145/190347.190371.

Kaltofen, Erich et al. (2008). "Approximate factorization of multivariate polynomials using singular value decomposition". In: *Journal of Symbolic Computation* 43.5, pp. 359 –376. ISSN: 0747-7171. DOI: http://dx.doi.org/10.1016/j.jsc.2007.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0747717107001538.

Kellerer, Hans, Ulrich Pferschy, and David Pisinger (2004). *Knapsack problems*. 1st. Springer Berlin Heidelberg. ISBN: 978-3-540-24777-7. DOI: 10.1007/978-3-540-24777-7.

Kellerer, Hans, Ulrich Pferschy, and Maria Grazia Speranza (1997a). "Algorithms and Computation: 8th International Symposium, ISAAC '97 Singapore, December 17–19, 1997 Proceedings". In: ed. by Hon Wai Leong, Hiroshi Imai, and Sanjay Jain. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. An efficient approximation scheme for the subset-sum problem, pp. 394–403. ISBN: 978-3-540-69662-9. DOI: 10.1007/3-540-63890-3_42. URL: http://dx.doi.org/10.1007/3-540-63890-3_42.

— (1997b). "An efficient approximation scheme for the subset-sum problem". English. In: *Algorithms and Computation*. Ed. by HonWai Leong, Hiroshi Imai, and Sanjay Jain. Vol. 1350. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 394–403. ISBN: 978-3-540-63890-2. DOI: 10.1007/3-540-63890-3_42. URL: http://dx.doi.org/10.1007/3-540-63890-3_42.

Kesh, Deepanjan and Shashank Mehta (2008). "Polynomial Irreducibility Testing Through Minkowski Summand Computation". In: *Proc. Canada Conf. Comp. Geometry*, pp. 43–46. URL: http://cccg.ca/proceedings/2008/paper10.pdf.

Khanduja, Sudesh K. and Jayanti Saha (1997). "On a generalization of Eisenstein's irreducibility criterion". In: *Mathematika* 44 (01), pp. 37–41. ISSN: 2041-7942. DOI: 10.1112/S0025579300011931. URL: http://journals.cambridge.org/article_S0025579300011931.

Krasauskas, Rimvydas and Ron Goldman (2003). "Toric Bezier Patches with Depth". In: *Topics in Algebraic Geometry and Geometric Modeling*. DOI: http://dx.doi.org/10.1090/conm/334.

Kulik, Ariel and Hadas Shachnai (2010). "There is No EPTAS for Two-dimensional Knapsack". In: *Inf. Process. Lett.* 110.16, pp. 707–710. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2010.05.031. URL: http://dx.doi.org/10.1016/j.ipl.2010.05.031.

Lenstra, A. K., H. W. Lenstra, and L. Lovász (1982). "Factoring polynomials with rational coefficients". In: *Mathematische Annalen* 261.4, pp. 515–534. ISSN: 1432-1807. DOI: 10.1007/BF01457454. URL: http://dx.doi.org/10.1007/BF01457454.

Maclagan, Diane and Bernd Sturmfels (2015). *Introduction to Tropical Geometry*. Vol. 161. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, pp. vii+359.

Magazine, Michael J. and Maw-Sheng Chern (1984). "A Note on Approximation Schemes for Multidimensional Knapsack Problems". In: *Math. Oper. Res.* 9.2, pp. 244–247. ISSN: 0364-765X. DOI: 10.1287/moor.9.2.244. URL: http://dx.doi.org/10.1287/moor.9.2.244.

McCreath, Eric C. (2008). "Partial Matching of Planar Polygons Under Translation and Rotation." In: *Proceedings of the 20th Canadian Conference on Computational Geometry*. Pp. 47–50.

Moses, J. and D. Y. Y. Yun (1973). "The EZGCD algorithm". In: *Proc. 1973 ACM National Conf.*

Murty, M. Ram (2002). "Prime Numbers and Irreducible Polynomials". In: *The American Mathematical Monthly* 109.5, pp. 452–458. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2695645.

Noro, Masayuki and Kazuhiro Yokoyama (2002). "Yet Another Practical Implementation of Polynomial Factorization over Finite Fields". In: *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*. ISSAC '02. Lille, France: ACM, pp. 200–206. ISBN: 1-58113-484-3. DOI: 10.1145/780506.780532. URL: http://doi.acm.org/10.1145/780506.780532.

Ostrowski, A. M. (1976). "On multiplication and factorization of polynomials, II. Irreducibility discussion". In: *aequationes mathematicae* 14.1, pp. 1–31. ISSN: 1420-8903. DOI: 10.1007/BF01836201. URL: http://dx.doi.org/10.1007/BF01836201.

Rabin, Michael O. and Michael O. Rabin (1979). "Probabilistic Algorithms In Finite Fields". In: *SIAM J. Comput* 9, pp. 273–280.

Salem, F. A., S. Gao, and A.G.B. Lauder (2004). "Factoring polynomials via polytopes". In: *In Proc. Intern. Symp. Symbolic and Algebraic Computation ISSAC*, pp. 4–

11. DOI: 10.1145/1005285.1005289. URL: http://doi.acm.org/10.1145/1005285.1005289.

Salem, Fatima Khaleb Abu (2004). "Factorisation Algorithms for Univariate and Bivariate Polynomials over Finite Fields".

Sasaki, Tateaki and Mutsuko Sasaki (1993). "A unified method for multivariate polynomial factorizations". In: *Japan Journal of Industrial and Applied Mathematics* 10.1, pp. 21–39. ISSN: 1868-937X. DOI: 10.1007/BF03167201. URL: http://dx.doi.org/10.1007/BF03167201.

Trager, Barry M. (1976). "Algebraic Factoring and Rational Function Integration". In: *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation*. SYMSAC '76. Yorktown Heights, New York, USA: ACM, pp. 219–226. DOI: 10.1145/800205.806338. URL: http://doi.acm.org/10.1145/800205.806338.

Vazirani, Vijay V. (2001). *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc. ISBN: 3-540-65367-8.

Veltkamp, Remco C. (2001). "Shape Matching: Similarity Measures and Algorithms". In: pp. 188–197.

Veltkamp, Remco C. and Michiel Hagedoorn (2001). "Principles of Visual Information Retrieval". In: ed. by Michael S. Lew. London, UK, UK: Springer-Verlag. Chap. State of the Art in Shape Matching, pp. 87–119. ISBN: 1-85233-381-2. URL: http://dl.acm.org/citation.cfm?id=370792.370810.

Wang, Paul S. (1976). "Factoring Larger Multivariate Polynomials". In: *SIGSAM Bull.* 10.4, pp. 42–42. ISSN: 0163-5824. DOI: 10.1145/1088222.1088227. URL: http://doi.acm.org/10.1145/1088222.1088227.

Weinberger, P. J. and L. P. Rothschild (1976). "Factoring Polynomials Over Algebraic Number Fields". In: *ACM Trans. Math. Softw.* 2.4, pp. 335–350. ISSN: 0098-3500. DOI: 10.1145/355705.355709. URL: http://doi.acm.org/10.1145/355705.355709.

Zassenhaus, Hans (1969). "On Hensel factorization, I". In: *Journal of Number Theory* 1.3, pp. 291 –311. ISSN: 0022-314X. DOI: http://dx.doi.org/10.1016/0022-314X(69)90047-X. URL: http://www.sciencedirect.com/science/article/pii/0022314X6990047X.

Zirdelis, Georgios (2014). "Manuscript on Minkowski decomposition of convex lattice polygons. Course Project".